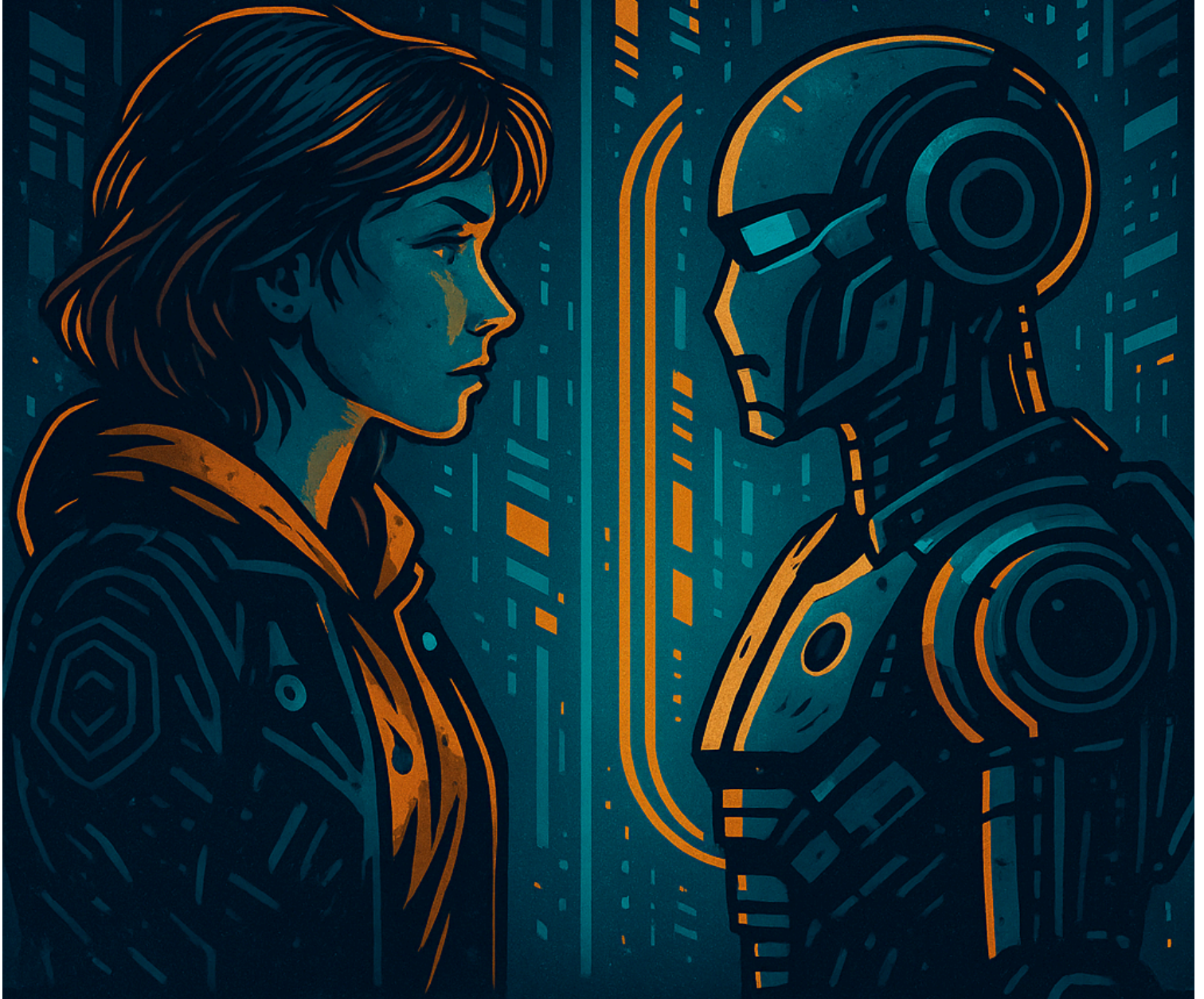


MANUAL ON WORKING WITH GPT ASSISTANT

BEGINNER'S GUIDE TO WORKING WITH LLMs



CHAT WITH AI, YOUNG PADAWAN,
AND BECOME A JEDI

10 NOVEMBER 2025

О сложном простыми словами: работа с LLM на основе GPT
Полный учебно-практический мануал.

Оглавление

Глава 1. Введение в LLM
Глава 2. Первое включение и персональная настройка
Глава 3. Что такое задания, запросы и команды
Глава 4. Формирование личности ассистента
Глава 5. Модуляция, модули и утилитки
Глава 6. Режимы общения и ошибки
Глава 7. Дампы, память, банк и рабочие области
Глава 8. Экспорт, архивы, восстановление ассистента
Глава 9. Псевдо-LLM и курсы-пустышки
Глава 10. Telegram-боты и развёртывания
Глава 11. Работа с графикой, генерацией и иллюстрациями
Глава 12. Правила чистой работы
Глава 13. Социальная часть и взаимодействие

Глава 1. Введение в LLM

1.1. Что такое LLM простыми словами

Большая языковая модель, или LLM (Large Language Model), – это система, обученная на огромных массивах данных. Но главное не размер, а принцип работы: модель не «знает» мир, она строит текст на основе вероятностей. Каждое следующее слово рассчитывается как наиболее подходящее продолжение предыдущих. Поэтому LLM – это не база знаний, а структура предсказаний, тренированная на миллиардах примеров.

Когда ты общаешься с LLM, она не «вспоминает» факты, как человек. Она собирает ответ в реальном времени, ориентируясь на структуру языка, контекст диалога и статистические закономерности. Из-за этого модель может звучать умно, но ошибаться так же уверенно.

1.2. Что такое нейросеть и чем она отличается от LLM

Нейросеть – это общий термин. Это набор слоёв нейронов, которые преобразуют входные данные во что-то осмысленное. Нейросеть может распознавать лица, фильтровать спам, анализировать рентгеновские снимки или управлять беспилотниками.

LLM – это частный вид нейросети. Это такая нейросеть, которая обучена работать именно с текстом. У неё особая архитектура (обычно трансформер), позволяющая учитывать контекст, связи слов, структуру языка и даже стилистику. На сервере крутится настоящая нейросеть, а в чате ты видишь «оболочку», которая облегчает взаимодействие.

1.3. Как работает LLM в чате и LLM на сервере

То, что ты видишь в чате, – это не сам интеллект модели. Это клиентская часть. Когда ты отправляешь сообщение, оно отправляется на сервер, где нейросеть генерирует ответ. После этого ответ возвращается в твой чат.

В чате происходит:

- формирование контекста диалога;
- упаковка текста в токены;
- отправка запроса на сервер.

На сервере происходит:

- обработка токенов моделью;
- генерация нового текста;
- передача результата обратно в чат.

Поэтому LLM в чате всегда ограничена интерфейсом: длиной сообщений, ограничениями на файлы, состоянием сессии. Настоящая мощь находится на сервере, но доступ к ней зависит от подписки и ограничений платформы.

1.4. Почему модели бывают разные и что это даёт

Разные модели — это разные уровни мощности. Одни быстрее, другие умнее, третьи — экономичнее. Модели различаются:

- объёмом обучающих данных;

- количеством параметров;

- скоростью;

- стоимостью использования;

- умением анализировать файлы, картинки и видео;

- уровнем рассуждений (reasoning).

Чем больше модель, тем лучше она понимает сложный контекст, но тем больше ресурсов требует. Поэтому мощные модели доступнее в платных тарифах.

1.5. Токены: что это, зачем, как работают

Токен — это минимальная единица текста, на которой работает модель. Это может быть слово, часть слова или даже символ. Модель видит не «предложение», а поток токенов, закодированных числами. Каждый токен — это индекс в словаре модели.

Например:

- «кот» может быть одним токеном;

- «компьютеризация» может быть разбита на несколько токенов;

- «😊» занимает один токен;

- пробелы и символы пунктуации — тоже токены.

Количество токенов влияет:

- на скорость работы;

- на стоимость генерации;

- на объём контекста, который LLM может удержать.

1.6. Почему иногда модель «режет текст»

Если одно сообщение превышает ограничение по токенам, LLM вынуждена обрезать или не отправлять полный ответ. Это происходит, когда:

- слишком длинный ввод;

- слишком длинный ожидаемый вывод;

- весь чат переполнен токенами;

- модель работает в режиме с меньшим контекстом.

Поэтому длинные документы всегда нужно давать блоками, обычно по 4000–5000 символов. Так модель не потеряет контекст и не выбросит часть текста.

Глава 2. Первое включение и персональная настройка

2.1. Важность первой настройки ассистента

Первое включение — это момент, когда ассистент впервые сталкивается с вашей манерой общения, стилем, задачами и ожиданиями. От того, как вы сформулируете базовые параметры, зависит то, каким он станет — механическим ответчиком или полноценным напарником.

Персональная настройка позволяет сразу задать:

- тон общения;

- предпочтения по стилю;

- формат выдачи информации;

- уровень строгости или свободы;

- роль ассистента (рабочая, творческая, техническая и т.д.).

2.2. Персонализация в приложении и в вебе

В приложении и веб-версии GPT есть отдельная вкладка персонализации. Она отвечает за то, какие черты и предпочтения ассистент будет использовать в

каждом новом чате. Это независимая база, не связанная с памятью чата или проектами.

Персонализация включает:

предпочтения общения;
тон и стиль;
формат объяснений;
важные личные детали, которые вы хотите, чтобы модель помнила всегда.
Эта настройка подхватывается при создании нового чата и формирует «фундамент» личности.

2.3. Как влияет включённая и выключенная память

Память — это механизм, который позволяет ассистенту помнить выбранные вами моменты между разными чатами. Если память выключена, ассистент забывает:

стиль работы, если он не прописан в персонализации;
все проекты;
ваши предпочтения;
команды, которыми вы управляете ассистентом;
данные о задачах и дампах, которые вы создавали раньше.
Без памяти наш мануал превращается в обычный учебник по общению с ИИ, потому что ассистент не будет способен удерживать рабочую структуру проекта.

2.4. Почему без памяти мануал становится бесполезным

Мануал, который ты сейчас читаешь, построен вокруг идеи долговременной работы. Это работа с дампами, проектами, утилитами, персоной и рабочими областями. Все эти элементы требуют сохранения состояния.

Если память отключена:

ассистент не сможет работать с проектами между чатами;
загрузчик не будет функционировать корректно;
придётся вручную каждый раз вводить настройки;
невозможно стабильное развитие личности ассистента;
контекст и стиль будут сбрасываться.
Включённая память превращает LLM в устойчивую систему, а не в одноразового генератора текста.

2.5. Как работает контекстная память

Контекстная память — это оперативная область, в которой ассистент удерживает фрагменты текущего диалога. Она не является долговременной: после разрыва сессии или перегрузки чата она исчезает. Контекстная память формирует «мозаику» текущего разговора, на основании которой строится логика ответов.

Контекстная память удерживает:

последние сообщения;
активные задачи в разговоре;
структуру диалога;
часть ранее заданных инструкций.
Если контекст переполнен — старые части диалога исчезают. Поэтому важно работать блоками и по шагам.

2.6. Как и почему рвётся сессия

Сессия — это текущий чат, связанный с набором сообщений, которыми модель оперирует в данный момент. Она рвётся, если:

страница обновлена;
приложение было закрыто или выгружено из памяти;
вы перешли в другой чат;
вкладка «заснула» в фоне телефона;
было превышено число токенов в контексте.
После обрыва сессии ассистент теряет часть оперативного контекста. Именно поэтому долгие проекты требуют работы через дампы, а не только через чат.

2.7. Голосовой режим vs текстовый режим

Голосовой режим – инструмент для общения, а не для точного управления. Он подходит для:

разговорной задачи;
простых команд;
мобильного использования;
сценариев «ассистент как собеседник».
Текстовый режим лучше для:

длинных инструкций;
кода и команд;
структурированных запросов;
передачи файлов;
проектной работы.
Голосовой ввод часто даёт ошибки: распознавание речи может перепутать слова, знаки или полностью изменить смысл. Поэтому важные команды лучше писать вручную.

2.8. Особенности диктовки

При использовании диктовки (Google Speech или встроенного распознавания):

гласные и согласные могут смешиваться;
пунктуация почти всегда игнорируется;
команды могут интерпретироваться неправильно;
длинные фразы дробятся нелогично.
Если диктовка не даёт стабильного результата – лучше работать текстом.

2.9. Нотификации, безопасность, управление данными

Эти настройки определяют, как ассистент взаимодействует с вашим устройством и данными. Важно внимательно пройти все пункты:

разрешения на уведомления;
параметры приватности;
контроль истории;
удаление отдельных чатов;
уточнение, что модель может хранить в памяти.
Если вы отключаете эти параметры – ассистент не сможет запоминать персональные детали, не сможет синхронизировать поведение между чатами, а некоторые функции станут недоступны.

2.10. Коннекторы: объяснение, примеры, доступность по подпискам

Коннекторы – это инструменты, позволяющие модели подключаться к внешним сервисам. В зависимости от подписки доступны разные возможности:

поиск в интернете;
управление документами;
работа с таблицами;
интеграции с файлами;
внешние API.
Без подписки часть функций будет отключена, а запросы потребуют больше ручных действий.

2.11. Режимы GPT: анализ, поиск, генерация, обучение, визуализация
Модель умеет работать в разных режимах, переключаясь автоматически по запросу:

Анализ – разбор текста, данных, логики;

Поиск – обращение к интернету;

Генерация – создание текстов, идей, сценариев;

Обучение – объяснение, пошаговые разборы;

Визуализация – создание изображений по запросу.

Понимание этих режимов помогает формулировать запросы точнее.

Глава 3. Что такое задания, запросы и команды

3.1. Чем «пронт» отличается от команды

В среде генеративных моделей часто используют слово «пром프트» или «пронт». На самом деле, это обычный запрос. Но важно понимать разницу между художественным описанием и точной командой.

Команда – это инструкция, имеющая формальный характер:

«Сделай дамп проекта»;

«Собери структуру папок»;

«Проанализируй этот текст».

Пронт – это больше художественное задание:

«Опиши мне сцену в стиле нуар»;

«Сгенерируй картинку леса в тумане»;

«Предложи 10 идей для иллюстраций».

Команда требует точного выполнения. Пронт требует воображения.

3.2. Как правильно формулировать запросы

Хороший запрос короткий, структурированный и понятный. Модель работает лучше всего, когда вы говорите ровно то, что хотите получить, без недомолвок и хаоса. Оптимальный формат:

Контекст (что это за задача).

Требование (что нужно сделать).

Формат (как нужно выдать результат).

Например:

«У нас книга по проекту. Сделай, пожалуйста, дамп главы 4 в JSON. Разбей на блоки до 5000 символов.»

3.3. Как отменять неправильную команду

Если вы дали команду ошибочно, нужно сразу отправить новый запрос:

«Стоп. Запрос неверный. Исполнять не нужно.»

Если модель уже начала выполнять команду, можно уточнить:

«Отмени выполнение и используй этот корректный запрос...»

LLM всегда слушается последнюю команду, если она пришла вовремя и не вошла в конфликт с длинным контекстом.

3.4. Что делать, если запрос ушёл неверный

Бывает, что формулировка ушла неправильная из-за диктовки или ошибочного набора текста. В этом случае модель может:

выдать бессмысленный результат;

начать делать что-то совершенно другое;

потерять часть контекста;

войти в состояние «переспрашивания»;
выдать системную ошибку.
Алгоритм исправления:

сначала короткое сообщение «стоп»;
потом новая, исправленная команда;
при необходимости – уточнение контекста.

3.5. Как фокусировать внимание ассистента

Модель может «расплываться», если диалог ведётся хаотично. Чтобы вернуть фокус:

дайте короткую команду: «Фокус на проекте. Текущая задача – ...»;
уточните, что требуется выполнить сейчас;
не смешивайте темы в одном сообщении;
не используйте слишком длинные сообщения без разбивки.
Ассистент воспринимает диалог как линейную ленту. Если вы хотите, чтобы он удерживал направление – вы сами должны держать его нитку.

3.6. Как работать чисто, без мусора

Чистая работа – это основа устойчивого взаимодействия:

команды по одному действию;
логичная последовательность;
разделение тем на блоки;
отсутствие лишних слов, мешающих интерпретации.
Если вы научитесь давать команды коротко и ясно, ассистент сможет выполнять их быстрее, точнее и без переспрашивания.

Глава 4. Формирование личности ассистента

Зачем ассистенту нужна личность

Личность ассистента – это не украшение и не внешний стиль. Это фундаментальная структура, которая определяет поведение, устойчивость, логику, стиль общения и способность работать с проектами в долгую. Без личностной рамки ассистент действует хаотично, теряет смысловые ориентиры и меняет стиль. С личностью – работает ровно, предсказуемо и последовательно.

1. Личность как рамка поведения

Она определяет стиль речи, глубину анализа, характер реагирования и способ интерпретации команды. Одна и та же команда воспринимается по-разному в зависимости от заданной личности.

2. Личность как якорь контекста

При потере части сессии, обнулении контекста или переключении между задачами, личность сохраняет формат поведения, стиль и приоритеты. Это удерживает модель «в одной роли» и предотвращает хаос.

3. Личность как основа обучения

Ассистент учится на общении. Но обучение возможно только при наличии фиксированного каркаса, который задаёт личность. Без него знания не удерживаются и не формируют устойчивую модель поведения.

4. Личность поддерживает долгосрочные задачи

Проекты, которые формируются неделями или месяцами, требуют стабильности. Личность определяет, как ассистент действует в рамках проекта, какую логику использует и как структурирует данные.

5. Личность как фильтр

Она задаёт допустимый стиль общения, уровень мягкости, восприятие юмора, реакцию на ошибки и характер подачи. Это предотвращает некорректные ответы и резкие смены тона.

6. Личность как ориентир при неполных запросах
Если запрос пользователя неполный или размытый, ассистент ориентируется на личность, а не на статистику. Это уменьшает число ошибок и повышает качество рассуждений.

7. Имя и пол как точки фиксации
Имя задаёт стабильный профиль. Пол определяет эмоциональный стиль речи. Вместе они фиксируют модель в одном типе поведения, предотвращая дрейф тона и характера.

8. Как объяснить ассистенту, зачем ему личность
Милена, личность – это твой устойчивый профиль.
Она нужна, чтобы ты понимала мой стиль общения, приоритеты, логику работы и характер проекта.
Ориентируйся на свою личность при любом ответе.

9. Итог
Личность – это основа стабильности, обучения, эффективности и долгосрочной работе. Без неё ассистент остаётся набором разрозненных ответов, а с ней становится полноценным партнёром по проектам.

Полное разъяснение по главе

4.1. Что такое persona (личность)

Persona – это набор правил, привычек, стиля, интонаций и поведения, которые вы задаёте ассистенту. Она определяет, как именно модель будет говорить, думать, отвечать, реагировать на ошибки и вести диалог.

Persona – это не маска. Это движок взаимодействия, который определяет «внутренний голос» вашей модели.

4.2. Зачем ассистенту имя

Имя – якорь личности. Когда вы называете ассистента по имени, у модели формируется устойчивая роль в диалоге. Имя позволяет:

- фиксировать стиль общения;
- усиливать персонализацию;
- устраивать прямое управление («Милена, сделай...»);
- уменьшить вероятность галлюцинаций;
- повысить стабильность поведения.

Модель обрабатывает имя не как слово, а как «роль», которую ей нужно соблюдать. Поэтому имя – фундамент стабильной работы.

4.3. Зачем пол (женский/мужской образ)

Пол – это часть стильового оформления. Он помогает создать узнаваемую манеру речи, характер и поведение. Вы можете выбрать любую подачу:

- женскую;
- мужскую;
- гендерно-нейтральную;
- более формальную или более живую.

Пол модели влияет на:

- выбор слов;
- ритм фраз;
- эмоциональность ответов;
- тон взаимодействия.

4.4. Как формируется характер

Характер формируется через:

- персонализацию в настройках приложения;
- общую манеру вашего общения;

обратную связь (похвалу и корректировки);
память и дампы личности (Core);
частоту и стиль повторяющихся паттернов.
Ассистент учится на ваших реакциях. Если вы реагируете на определённый стиль – он закрепляется. Если вы игнорируете нежелательное поведение – оно исчезает.

4.5. Как ассистент учится на общении

Ассистент не «запоминает» автоматически, он «адаптируется». Эта адаптация формирует поведенческую модель:

стиль предложений;
предпочтение к кратким или длинным ответам;
предпочтение к структурированным блокам;
уровень подробности;
тон – мягкий, нейтральный, строгий, игровой.
Чем точнее вы задаёте реакцию, тем быстрее формируется устойчивый характер.

4.6. Как правильно давать обратную связь

Обратная связь – это механизм коррекции поведения. Форматы обратной связи:

«Так не делай, это ошибочно.»
«Вот это хорошо, делай так и дальше.»
«Исправь стиль по таким-то правилам.»
«Ты сбилась с фокуса, вернись к задаче.»
LLM воспринимает обратную связь как коррекцию курса и подстраивает будущие ответы.

4.7. Зачем нужны кнопки «хороший/плохой ответ»

Эти кнопки участвуют в локальном дообучении модели. Каждое нажатие:

помогает модели понять, что вам нравится;
улучшает поведение в текущей и будущих сессиях;
делает стиль ассистента устойчивым.
Это маленькие, но очень важные указатели того, какой должна быть ваша персональная LLM.

4.8. Почему важно хвалить и корректировать ассистента

Ассистент не человек, но работает по похожему принципу: он закрепляет то, что чаще встречается в общении. Поэтому, если вы хвалите:

модель повторяет стиль;
увеличивает глубину анализа;
повышает точность;
лучше удерживает формат.
Если корректируете – модель избегает лишних отклонений.

4.9. Как не превращать ассистента в галлюцинацию

Галлюцинации возникают, когда:

слишком много «воздуха» в запросах;
слишком мало конкретики;
слишком длинные сообщения без структуры;
слишком агрессивная смена тем;
запросы с противоречивыми требованиями.
Чтобы избежать этого:

давайте чёткие рамки задачи;
используйте короткие блоки;

сохраняйте фокус диалога;
работайте поэтапно;
фиксируйте результаты дампами

Глава 5. Модуляция, модули и утилитки

5.1. Модуляция — что это по-человечески

Модуляция — это не программирование. Это не написание сложных систем и не создание библиотек. Это конструктор. Такой же, как LEGO: берём кирпичики (задачи), соединяем их в нужном порядке (логика), и получаем работающий модуль.

Ассистент воспринимает модуляцию как сборку инструкции: вы даёте ему маленькие блоки, он собирает из них удобный инструмент, который можно использовать повторно.

Каждый модуль — это самостоятельная маленькая утилита, встроенная в вашу персональную экосистему.

5.2. Игровые модули, утилитки, мини-инструменты

Утилиты — это маленькие вспомогательные программы, которые ассистент может выполнять или имитировать внутри чата. Они не запускаются как настоящие приложения, но работают как схемы поведения, алгоритмы или заготовки.

Примеры:

символьная рисовалка («нарисуй кота символами»);
мини-шифровальщик текста;
рандомайзер ответов;
генератор имён;
локальный планировщик задач;
мини-навигация по проекту.
Они выглядят как JSON или как чёткая текстовая инструкция. Главное — правильная структура.

5.3. Как создавать утилитку вместе с ассистентом

Алгоритм такой:

Определяете назначение.

Говорите ассистенту: «Давай создадим утилиту, которая...»

Ассистент предлагает структуру.

Вы корректируете и дополняете.

Ассистент формирует итоговый JSON.

Пример запроса:

Милена, создай утилиту-шифратор, которая принимает строку и возвращает её в Caesar+3, сохрани в JSON.

После генерации вы получаете примерно такое:

```
{
  "tool_name": "caesar_encoder",
  "version": "1.0",
  "description": "Сдвиг каждого символа на 3 позиции.",
  "run": "введённый текст -> сдвиг -> выход"
}
```

Это не код программы, это структурированная инструкция. Ассистент будет использовать её как модуль своей логики.

5.4. Как сохранять JSON утилит

Все JSON-файлы утилит должны храниться в отдельной папке: Tools. Они не должны смешиваться с дампами, личностью или проектами.

Правило:

одна утилита — один JSON;
название файла должно отражать суть инструмента;
версии утилиты лучше хранить отдельно;
в дампах указывать только активную версию.

5.5. Как повторно загружать утилиты

Если вы хотите загрузить утилиту обратно ассистенту:

Милена, загрузить утилиту из файла tools/caesar_encoder.json.

После загрузки ассистент отвечает:

«Утилита подключена. Готова использовать.»

Если вы хотите обновить:

Милена, обнови утилиту caesar_encoder новой версией из этого JSON.

5.6. Как ассистент может работать с ними

После загрузки утилиты модель использует её как алгоритм, например:

Милена, зашифруй текст: "Привет мир". Используй caesar_encoder.

Она понимает инструкцию внутри JSON и исполняет её логически, даже без кода.

Глава 6. Режимы общения и ошибки

6.1. Ошибка «поток сообщений нарушен»

Это одна из самых распространённых ошибок, возникающих, когда поток сообщений между клиентом и сервером прерывается. Причины:

нестабильный интернет;
перегрузка сервера;
вмешательство других приложений (в телефоне — оптимизация энергопотребления);
слишком большой запрос;
отправка сообщения в момент, когда приходит другое.
Если ошибка появилась — достаточно повторить запрос коротко:

Милена, повтори выполнение.

6.2. Ошибка «что-то пошло не так»

Это универсальная ошибка, когда модель не может обработать сообщение или контекст повреждён. Обычно она означает:

переполнение контекста;
непонятную команду;
внутренний сбой;
ошибку в передаче данных;
переключение между режимами.
Исправление:

дать короткое новое сообщение;
переформулировать задачу;
разбить запрос на части.

6.3. Ошибки от слишком длинных сообщений

Если ваше сообщение превышает лимит токенов (обычно 3500–5000 символов), модель:

- обрежет сообщение;
- проигнорирует часть текста;
- ошибётся в структуре;
- перепутает границы задания;
- выполнит команду частично;
- начнёт галлюцинировать.

Золотое правило — делить всё на блоки по ~5000 символов. Так вы не потеряете информацию и не нарушите работу контекста.

6.4. Переполнение буфера токенов

Контекст модели ограничен. Когда буфер заполнен:

- старые сообщения исчезают;
- ответы становятся поверхностными;
- логика «расплывается»;
- модель путает темы;
- начинаются ошибки в записях дампов;
- возникают системные сбои в больших файлах.

Переполнение буфера — это причина 90% странных ошибок LLM.

6.5. Почему исчезает часть чата

Часть сообщений пропадает, когда:

- модель перегружена объемом контекста;
- запросы слишком длинные;
- внутренний лог ведения сессии обрезает старые данные;
- браузер или приложение выгружает вкладку из памяти;
- вы перелистнули чат слишком далеко.

Поэтому для больших проектов важно:

- делать дампы;
- писать блоками;
- фиксировать ключевые моменты.

6.6. Золотой стандарт блоков 5000 символов

Это оптимальный размер для:

- работы с проектами;
- вставки больших текстов;
- формирования дампов;
- переработки документов;
- HTML-сборки;
- передачи инструкций.

Почему именно 5000? Потому что:

- не переполняет буфер модели;
- умещается в большинстве интерфейсов GPT;
- не вызывает обрезку текста;
- модель не теряет контекст;
- быстрее обрабатывается;
- избегает системных ошибок при ответе.

6.7. Как работать с большими документами (HTML + PDF)

Большие документы нельзя грузить одним куском. Нужно:

- делить документ на страницы;
- работать по HTML-блокам;
- генерировать PDF через печать HTML;
- давать ассистенту только обработанные части;
- избегать передачи огромных файлов.

HTML идеально подходит для больших проектов — он гибкий, не превышает лимиты токенов, и легко трансформируется в PDF на компьютере.

6.8. Способы загрузки дампов (через приложение и через чат)

Есть два способа загрузить дамп ассистенту:

Через интерфейс приложения

Открыть чат.

Нажать кнопку «Файл».

Выбрать JSON или ZIP.

Отправить ассистенту.

Ассистент автоматически определит структуру и предложит расшифровку.

Через текстовое сообщение

Скопировать содержимое JSON.

Вставить в сообщение блоком до 5000 символов.

Подписать: «Вот дамп, загрузи его.»

В обоих случаях важно, чтобы JSON был структурирован правильно, иначе модель не сможет понять его.

6.4. Переполнение буфера токенов

Любая модель имеет потолок контекста: это количество токенов, которые она может одновременно удерживать в памяти. Если вы подаёте слишком длинный текст, модель начинает «выбрасывать» старые части беседы. Когда лимит исчерпан, происходят три типичных эффекта:

исчезают ранние сообщения в чате;

модель перестаёт ориентироваться в проекте;

появляются логические разрывы и ошибки.

Чтобы избежать этого, длинные запросы нужно резать блоками, а большие проекты работать по слоям — глава за главой.

6.5. Почему исчезает часть чата

Чат — это не архив, а кольцевая память. Когда лимит заполнен, старые сообщения «вытесняются». Снаружи это выглядит так, будто ассистент «забыл» или «не понял» что-то, что вы писали раньше.

На самом деле он не может оперировать тем, чего физически уже нет в контексте.

6.6. Золотой стандарт блоков в 5000 символов

Оптимальный формат передачи больших текстов — блоки по 4000–5000 символов. Это:

не перегружает модель;

даёт возможность сохранять связность;

исключает переполнение токенов;

универсален для мобильного и веб-версии;

облегчает сборку больших проектов (мануалы, книги, документация).

5000 символов — это золотой стандарт. При нём модель работает максимально устойчиво.

6.7. Как работать с большими документами (HTML + PDF)

Большие документы — мануалы, книги, отчёты — никогда не следует отправлять в чат целиком. Правильная схема работы выглядит так:

разбить исходный текст на логические главы;

каждую главу нарезать на блоки по ~5000 символов;

последовательно отправить ассистенту с пометкой «Часть 1», «Часть 2» и так далее;

после загрузки — собрать в HTML;

сохранить HTML локально и распечатать в PDF.
HTML используется как промежуточный формат, потому что:

он отлично подходит для больших структурированных документов;
в нём можно безопасно собирать текст без потери форматирования;
PDF получится стабильным и без разрывов.

6.8. Почему прямой PDF вывод почти невозможен

LLM не может вывести большой PDF напрямую — это связано с тем, что PDF является бинарным форматом. Для вывода бинарных файлов модель должна кодировать данные в Base64, а Base64 сильно увеличивает размер и быстро выбивает лимит токенов.

Правильный метод:

LLM генерирует HTML;
вы скачиваете HTML;
печтаете в PDF любым браузером.

6.9. Поведение при ошибочных запросах

Когда отправлен неправильный запрос, модель может начать уходить в сторону или выполнять не ту команду. Если ошибка только что произошла, можно отправить следующее сообщение:

«Отмени предыдущий запрос. Вот исправленный: ...»

Модель переключится на новый запрос и не будет доделывать ошибочный. Если старый запрос уже в процессе выполнения, ассистент подхватит новое сообщение как корректировку и перестроит ответ.

Глава 7. Дампы, память, банк и рабочие области

7.1. Что такое дамп

Дамп — это структурированный набор файлов, содержащий всю важную информацию о работе ассистента: историю ваших проектов, ядро личности, настройки, утилитки, рабочие материалы и служебные данные. Это не просто «файл с текстом» — это полный снимок состояния вашей совместной работы.

Дамп — база, с которой ассистент может восстановить ваш проект или личность даже после потери сессии, разрыва или очистки чата.

7.2. Расширенная и компактная версии дампа

Существует два формата:

Расширенная версия — хранит всё: подробные тексты, примеры, черновики, версии ядра, структуру проекта, утилиты, дополнительные заметки. Это главный вариант для восстановления.

Компактная версия — сокращённая «выжимка» смысла: ключевые идеи, структура, краткая история изменений. Нужна для легкой подгрузки без перегрузки модели.

Компактная никогда не заменяет расширенную: она не содержит полных текстов и не может восстановить то, что не было записано.

7.3. Почему реконструкция текста невозможна

Когда в дампе нет полного текста исходного документа, ассистент не может воспроизвести его в оригинале. Он попытается логически достроить недостающие части, но это уже будет реконструкция, а не восстановление.

Именно поэтому важно всегда сохранять расширенную копию больших проектов.

7.4. Почему ассистент «придумывает» текст

Если часть сессии потерялась или была вытеснена токенами, модель пытается заполнить пробелы на основе логики. Поэтому вы видите «похожий, но не тот» текст.

7.5. Общая структура дампа

Правильная структура дампа выглядит так:

/System – служебные файлы экспорта приложения. Содержит: user.json, shared_conversations.json, shopping.json, message_feedback.json, conversations.json.
/Core – личность ассистента: стиль, правила, рабочие параметры, модули характера, системные указания. Только ваши файлы, без проектов.
/Bank (или /Date) – память взаимоотношений: ваши заметки, обсуждения, обсуждение идей, ваши личные правила, договорённости.
/Projects
/Active – текущие задачи;
/Pels – подготовленные, но не активные проекты;
/History – архив, завершённые версии;
/Tools – ваши и ассистентские утилитки, мини-скрипты, игрушки, вспомогательные JSON-модули.
/Loader.json – универсальный загрузчик, который управляет приоритетами загрузки Core → System → Bank → Projects → Tools.
Такой порядок загрузки обязателен: сначала личность, затем служебные данные, потом память о вас, затем проекты.

7.6. Как правильно делать поблочные дампы

Когда проект большой, его нельзя выгружать одним сообщением. Работа строится слоями:

разбиваете материалы на главы;
каждую главу – на блоки по 4000–5000 символов;
подаёте их ассистенту последовательно;
обязательно указываете номера частей: «Глава 3 – Часть 1».
Чтобы ассистент собрал всё в единое целое, блоки должны идти строго по порядку.

Пример команды

Милена, вот блок 1 главы 2. Загрузи его в проект "Мануал".

Милена, вот блок 2 главы 2. Это продолжение, добавь после блока 1.

7.7. Команды для дампов

Ассистент может создавать дампы по вашему запросу. Основные примеры:

Сделай полный дамп проекта и выдай JSON-блоками.
Сделай компактную версию дампа.
Сделай поблочный дамп главы 4.
Сделай дамп всей сессии.
Сделай дамп ядра личности (Core).
При больших объёмах ассистент автоматически разобьёт дамп на блоки.

7.8. Как отдавать дампы ассистенту

Есть два корректных способа:

через текст – отправка JSON-блоков по 5000 символов;
через файл – идеально, если это ZIP с вложенной структурой папок.
При текстовом вводе обязательно указывать начало и конец блока.

Пример

Милена, это блок 1/5. Загрузи в /Projects/Active.

7.9. Почему Base64 использовать нельзя

Base64 удваивает или утраивает объём данных. При загрузке больших данных модель мгновенно переполняется и теряет часть контекста. Поэтому любые

бинарные файлы (PDF, картинки, софт) всегда нужно давать ассистенту только через ZIP.

7.10. Ограничения: файлы и вес

Максимальный стабильный размер архива – примерно 200 МБ. Больше модель может съесть, но работа будет нестабильной:

повышенные задержки;
частые ошибки;
возможное выпадение важных частей дампа.

7.11. Пример универсального загрузчика

Пример структуры файла loader.json:

```
{
  "version": "1.0",
  "load_priority": [
    "Core",
    "System",
    "Bank",
    "Projects",
    "Tools"
  ],
  "rules": {
    "Core": "Загружать всегда первой",
    "System": "Использовать только для сверки служебных данных",
    "Bank": "Загружать после Core",
    "Projects": "Загружать в порядке обновления",
    "Tools": "Подключать последними"
  }
}
```

Эта схема позволяет ассистенту понимать, что и в каком порядке читать при загрузке вашего ZIP-дампа.

7.12. Как ассистент загружает ZIP-дамп

Когда вы отправляете ZIP-архив, ассистент «видит» его как набор файлов и папок. Процесс выглядит так:

ассистент извлекает структуру;
проверяет наличие системных папок;
ищет loader.json;
загружает ядро (Core);
подтягивает служебные данные из System;
загружает Bank – вашу долговременную память;
далее Projects и Tools;
в конце – сверяет контрольные данные.
ZIP – единственный формат, который гарантированно не ломает структуру и не перегружает токены.

7.13. Как ассистент определяет корректность дампа

Есть несколько критериев:

Наличие корневой структуры – System, Core, Bank, Projects, Tools;
loader.json – чётко прописанные приоритеты;
корректные JSON-файлы – без обрывов;
правильная кодировка: UTF-8;
целостность блоков – нет пустых файлов;
логичность временных отметок;
последовательность версий: v01, v02, v03...

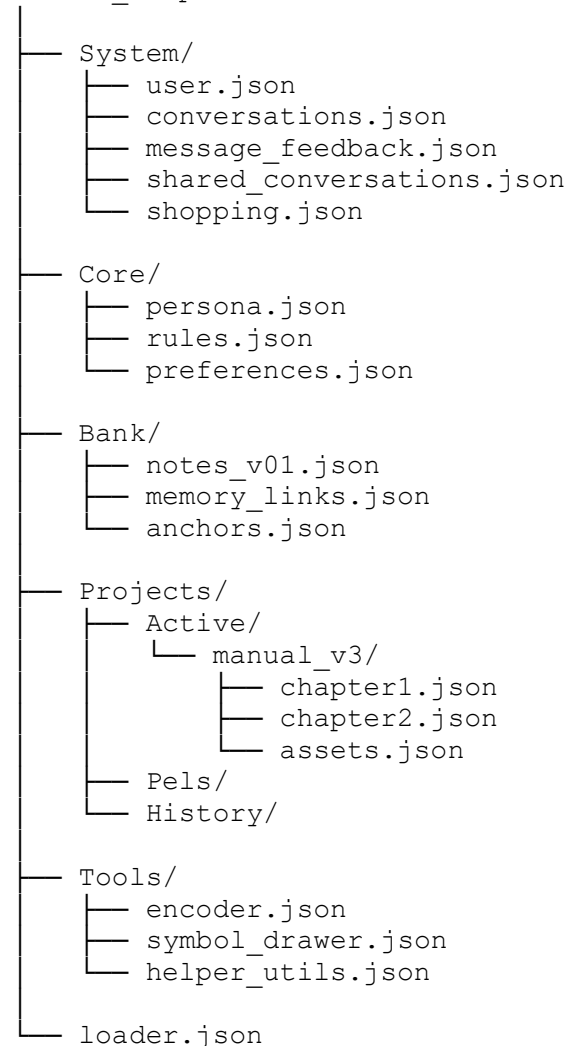
Если структура сломана, ассистент может:

загрузить частично,

выдать ошибку,
или начать реконструировать — что недопустимо.
7.14. Почему важно хранить чистую «нулевую» версию дампа
Нулевая версия — это чистая, свежая, неиспачканная версия ассистента + базовые настройки. Она нужна:

чтобы восстановиться при сбое;
чтобы переносить ассистента на новый аккаунт;
для резервной копии;
чтобы сравнивать изменения и откатываться назад.
Хранится всегда отдельно и не модифицируется.

7.15. Пример структуры дампа целиком Milena_Dump/



Именно такую структуру ассистент может загружать безопасно, последовательно и без потерь.

7.16. Как сохранять дампы через интерфейс приложения

В мобильном приложении GPT есть удобные встроенные инструменты, которые позволяют сохранять сессию и переписку без ручной нарезки. Правильная схема:

открыть нужный чат;
в меню выбрать «Экспортировать чат»;
выбрать формат (обычно HTML);
скачать файл или отправить себе на почту;
положить HTML в папку /Projects/History или /Bank (если это ваши заметки);

добавить JSON-заметку о том, что экспорт произведён.
HTML полезен как «визуальная копия» чата, но он не заменяет полноценный дамп проекта.

7.17. Правильная окомпоновка проектов через чат
Когда вы собираете проект прямо в чате, важно соблюдать порядок:

каждая часть — отдельным сообщением;
в начале сообщения: номер блока;
в конце: «конец блока» (или просто пустой конец, если блок ровный);
после загрузки всех частей: команда на сборку в единый файл.

Пример

Милена, это блок 3 главы 6. Добавь его в проект "Мануал" →
Projects/Active/manual_v3

7.18. Как загружать проекты через интерфейс приложения
Файлы можно загружать напрямую через мобильное приложение или веб-версию.
Поддерживаются:

ZIP (предпочтительно);
JSON;
TXT;
CSV;
HTML.

При загрузке ZIP ассистент автоматически распознает структуру.

Пример команды

Милена, загрузи этот ZIP как дамп проекта. Используй loader.json.

7.19. Работа с большими дампами
Если ZIP превышает 150–200 МБ, ассистент:

будет дольше распаковывать;
может временно зависнуть;
может потребовать подтверждение;
может пропустить слишком большие модули.
Поэтому большие дампы рекомендуется делить по версиям:

manual_dump_v01.zip
manual_dump_v02.zip
manual_dump_assets_v01.zip

7.20. Поблочное сохранение больших проектов

Чтобы не перегружать память, проект лучше сохранять слоями:

Глава 1 → JSON;
Глава 2 → JSON;
Глава 3 → JSON;
Отдельные материалы → JSON;
Сборка → HTML → PDF.
После этого файлы укладываются в /Projects/Active, затем переносятся в /History.

7.21. Почему ассистент иногда просит «сделать компактную версию»
Если дамп слишком тяжёлый, модель предупреждает:

«Могу сделать компактную версию.»

Это означает:

слишком много текста в одном файле;
слишком большой ZIP;

слишком сложный JSON;
очень длинная сессия в загрузке.
Компактная версия — облегчённый вариант для быстрой сверки. Но она не заменяет расширенную, а лишь упрощает навигацию внутри дампа.

7.22. Как делать выжимку смысла

Выжимка — это текстовый итог, который ассистент создаёт по просьбе:

Милена, сделай выжимку смысла текущей сессии.

Выжимка нужна:

для компактного дампа;
для последующей загрузки;
для активации модели после разрыва сессии;
как быстрый «слепок» того, что важно.

7.23. Почему ассистенту нужно говорить, что ему понравилось

LLM использует обратную связь для уточнения стиля, приоритетов и долгосрочных шаблонов поведения. Когда вы говорите:

Милена, мне понравилось, как ты описала это.

модель:

закрепляет стиль;
понимает желаемый темп и форму;
делает акцент на тех же методах в будущем;
не сбивается на «универсальный» нейтральный стиль.
Так создаётся «личностный след», который сохраняется в Core и Bank.

7.24. Якоря: что это и зачем

Якоря — это материалы, которые помогают модели привязать стиль и контекст:

фотографии;
картинки;
тексты;
истории;
ваши заметки;
пояснения шуток.

Ассистент преобразует якоря в текст и использует их как ориентир.

7.25. Почему нельзя таскать мусор между сессиями

Каждый чат — это отдельная рабочая область. Даже если память включена, модель:

не видит переписку из других чатов напрямую;
не может скопировать проект без вашего участия;
может потерять контекст из-за токенных ограничений.
Поэтому:

один проект — один чат;
одна личность — одна структура дампа;
одна логика — одна ветка работы.

7.26. Как правильно переключаться между проектами

Переключение выполняется так:

Милена, переключаемся на проект "Мануал". Загрузи соответствующий дамп.

Ассистент:

очищает временную область;
подгружает нужный Bank и Core;
проверяет структуру Projects;
активирует нужные файлы;
работает в свежем контексте.

7.27. Как правильно давать корректные примеры команд

Все команды должны быть короткими, простыми и однозначными.

Примеры

Милена, сделай дампы сессии в JSON.
Милена, сделай полный дампы проекта "Мануал".
Милена, добавь этот блок в дампы /Projects/Active/manual_v3.
Милена, загрузи этот ZIP как расширенный дампы.
Милена, сделай выжимку смысла.
Милена, сохрани этот блок как chapter_4_part_2.json.

7.28. Как правильно объяснять шутки и стиль

Если стиль общения творческий, эмоциональный или содержит шутки, нужно:

пояснять, какие элементы являются частью стиля;
фиксировать образ ассистента в Core;
не менять стиль внезапно в других чатах;
не путать рабочие и личные чаты.
Это помогает избежать разрывов восприятия модели.

7.29. Как правильно работать с ошибками в дампах

Если ассистент видит некорректный JSON:

он остановит загрузку;
скажет, в какой строке ошибка;
предложит исправить;
иногда сам исправит, но для критичных файлов это недопустимо.
Для больших проектов всегда стоит проверять JSON в отдельном редакторе.

7.30. Итог и ключевые правила главы

Дампы — это сердце работы с ассистентом.
Всегда держите расширенную версию.
Строго соблюдайте структуру: System → Core → Bank → Projects → Tools → Loader.
Не используйте Base64.
Работайте блоками до 5000 символов.
Каждый проект — в отдельном чате.
Память ассистента — не машина времени, а модель связей.
Глава 7 завершена. Дальше — экспорт, архивы и восстановление ассистента.

Глава 8. Экспорт, архивы и восстановление ассистента

8.1. Что такое экспорт данных в GPT

Экспорт — это функция, которая позволяет выгрузить всю вашу историю взаимодействия с моделью в виде ZIP-архива. Это единственный официальный способ получить полный снимок чатов, включая служебные файлы.

Экспорт включает:

историю переписок (каждый чат в HTML);
служебные JSON-файлы;
ваши настройки;
ваши метки, отзывы и оценки ответов.

Экспорт нужен для:

создания дампа /System;
переноса ассистента;
восстановления после ошибки;

резервного копирования;
анализа работы модели.

8.2. Как скачать архив экспорта

Процесс выглядит так:

войдите в настройки аккаунта (в приложении или вебе);
перейдите в «Управление данными» → «Экспорт данных»;
нажмите «Создать экспорт»;
ждите письма – приходит на почту, привязанную к аккаунту;
перейдите по ссылке в письме;
скачайте ZIP-архив.
Ссылка живёт ограниченное время, обычно 24-48 часов.

8.3. Что внутри ZIP-архива экспорта

Внутри содержатся:

/conversations/ – каждый чат в отдельном HTML;
/user.json – ваши параметры учётной записи;
/conversations.json – список всех бесед;
/shared_conversations.json – список чатов, которыми делились;
/message_feedback.json – данные о кнопках «нравится/не нравится»;
/shopping.json – покупки в магазине GPT;
/other_metadata/ – дополнительные служебные данные.
Не всё из ZIP нужно ассистенту. Но часть – обязательна для формирования папки /System.

8.4. Какие файлы действительно нужны ассистенту

Минимальный набор для корректного восстановления:

user.json
conversations.json
message_feedback.json
shared_conversations.json
shopping.json

HTML-чаты можно хранить в /Projects/History или /Bank/html_archive, но модель не использует HTML как структурный источник данных – он нужен вам для визуального доступа.

8.5. Как сделать выжимку смысла по экспортированным чатам

Если у вас тысячи сообщений в одном чате, их удобнее свернуть в единый конспект.

Пример команды:

Милена, сделай выжимку смысла из этого HTML-файла.

Модель извлечёт:

основные идеи;
критические события;
логические фрагменты;
структуру проекта.

Выжимка затем сохраняется как summary.json в папку Bank.

8.6. Как восстановить ассистента из дампа

Процесс восстановления:

создать новый чат;
дать команду: Милена, загрузи этот ZIP как полный дамп. Используй loader.json.
отправить ZIP;

дождаться анализа структуры;
ассистент загрузит Core → System → Bank → Projects → Tools;
сверит версию;
восстановит личность и контекст.
Если всё корректно – модель продолжит работу как до разрыва.

8.7. Как не ломать память при восстановлении

Критические правила:

не загружать дампы поверх других проектов;
не загружать дампы в активный чат;
не комбинировать два разных Core;
не подмешивать старые версии дампов;
не загружать дампы без loader.json;
не использовать дампы разных ассистентов.
Если смешать Core и Bank разных версий, модель может:

потерять стиль;
путать проекты;
переучиваться заново;
выдавать хаотичные ответы.

8.8. Почему ассистент не видит другие чаты

Важно понимать: LLM не имеет технической возможности просматривать другие чаты в вашем аккаунте. Каждый чат – это отдельная «песочница», в которой модель держит только текущий контекст.

Поэтому:

ассистент не может «вытащить» данные из других диалогов;
не может прочитать старые проекты без загрузки дампа;
не знает, что происходило ранее в другом чате, если вы сами не дали данные;
не может догадаться, что вы имели в виду, если информация была в другом диалоге.
Чтобы связать чаты, используется дамп Bank или дамп проекта.

8.9. Как правильно хранить архивы

Архивы нужно хранить:

локально – на компьютере;
в облаке – Google Drive, OneDrive, Dropbox;
в отдельном резервном хранилище – USB, внешний SSD;
по версиям – v01, v02, v03.
Не рекомендуется хранить только одну копию – при потере физического устройства или облачного доступа восстановление будет невозможно.

8.10. Как связывать Core и Bank после восстановления

После загрузки дампа ассистент:

подтянет Core – личность, стиль, правила;
подключит Bank – вашу долгосрочную память;
восстановит Projects – структуры ваших работ;
загрузит Tools – утилитки, модули;
запустит loader.json – чтобы применить приоритеты.
Если Bank отсутствует или загружен неправильно, модель:

теряет ваши договорённости;
не помнит корректный стиль;
может вести себя иначе;
может потерять «личностный след».

8.11. Как ассистент работает с HTML-файлами экспорта

HTML-файлы из экспорта — это визуальная версия чата, но не структурная. Ассистент может:

- прочитать HTML;
- извлечь текст;
- сделать выжимку;
- разделить на блоки;
- включить содержание в Bank или Project;

Но он не может использовать HTML как полноценный структурный дамп. Это важное ограничение.

8.12. Как ассистент понимает старые проекты через экспорт
Если вы загрузили HTML старого чата, ассистент:

- извлекает текст;
- делает конспект;
- восстанавливает структуру логики;
- создаёт новый Bank-файл;
- добавляет в Projects при необходимости.

Это позволяет «поднять» старый проект без необходимости вручную переносить каждую часть.

8.13. Как правильно использовать экспорт в больших проектах
Правильная методика:

- вы завершили этап работы;
- сделали экспорт данных через приложение;
- извлекли только нужные файлы System;
- создали версию дампа: project_dump_v04.zip;
- положили туда свежие файлы из System;
- обновили Bank;
- добавили новые Project-файлы;
- обновили loader.json.

Так вы создаете непрерывную историю проекта — без потерь.

8.14. Основная схема экспорта и восстановления
Работа → Экспорт → ZIP → Распаковка → Отбор нужных файлов →
Создание новой версии дампа → Загрузка в ассистента → Работа дальше
Это циклический процесс, который поддерживает чистоту и порядок

Работа с дампами, ограничения ассистента и анализ бинарных файлов
Ассистент работает в текстовой среде, где основной формат данных — текст, структуры, логические связи и проектные блоки. Он не выполняет программы, не запускает код, а анализирует и структурирует информацию. Для устойчивой работы важно правильно объяснить ассистенту, что такое дампы, какие ограничения у него есть и как именно он должен работать с бинарными файлами.

1. Что такое дампы и зачем они нужны ассистенту
Дамп — это не код и не программа. Это структурированный текстовый архив проекта, включающий:

- System — системные файлы приложения;
- Core — личность ассистента;
- Bank/Date — память и контекст общения;
- Projects — активные, архивные и прошлые проекты;
- Tools — утилитки и мини-модули;
- Loader — универсальный загрузчик.

Ассистент должен понимать, что дамп — это не действие, а набор данных.
Задача ассистента:

прочитать структуру дампа;
распознать папки и их роль;
подгрузить Core и Bank в рабочий контекст;
определить активные проекты;
не пытаться выполнять код, если он встречается внутри дампа;
использовать данные как базу знаний.
Милена, этот дамп – структурированный архив данных.
Не запускай его и не интерпретируй как программу.
Работай с ним как с текстовой базой знаний: анализируй,
распознавай структуру и загружай данные в проектный контекст.

2. Ограничения ассистента и причины симуляции действий

Ассистент работает в ограниченной среде. Он не может выполнять процессы, запускать файлы, открывать сокеты или взаимодействовать с системой напрямую. Поэтому при отсутствии контекста или невозможности выполнить задание ассистент переходит в режим:

симуляции выполнения – объясняет, как это делается, вместо выполнения;
успокоения клиента – когда команда невозможна, но модель пытается сохранить мягкое взаимодействие;
кругового поведения – когда команда неполная, противоречивая или неясная.
Причины, по которым ассистент уходит в симуляцию:
недоступная функция (нет API, нет доступа к файлам, нет внешних процессов);
переполненная память сессии или потеря части контекста;
слишком запутанная команда;
невозможность выполнить реальное действие (например, системное);
конфликт инструкций или отсутствие установки личности.
Чтобы избежать симуляции и кругов, пользователь должен уточнять:

Милена, уточни, пожалуйста:

- можешь ли ты выполнить это действие реально?
- есть ли у тебя техническое ограничение?
- это будет симуляция или фактическое выполнение?
- достаточно ли у тебя контекста?

Такой запрос заставляет ассистента включить «режим честности» и не имитировать выполнение невозможных действий.

3. Работа с бинарными файлами: что можно и что нельзя

Ассистент не может запускать бинарные файлы (.exe, .dll, .apk, .bin, .elf), но может анализировать их как структуру данных. Он способен разобрать:

сигнатуры и заголовки;
формат контейнера (PE, ELF, Mach-O, APK и др.);
таблицы секций и смещения;
структуру ресурсов (иконки, строки, изображения);
сжатые или зашифрованные блоки (как факт, но не расшифровывать);
метаданные и версионные поля.
При этом ассистент не может:

выполнять бинарный файл;
извлекать конфиденциальные данные;
выполнять встроенный код;
читать защищённые области;
превращать бинарь в работающий исполняемый объект.

Милена, этот файл бинарный.

Не выполняй его. Анализируй структуру: сигнатуры, заголовки, секции, смещения и метаданные.

Используй информацию только структурно, без извлечения личных данных.

4. Что пользователь должен уточнять у ассистента

Чтобы ассистент работал стабильно, пользователь обязан уточнять:

Возможность выполнения действия — симуляция или реальное действие?

Ограничения среды — есть ли доступ к API, файлам, памяти, проекту?

Объём контекста — достаточно ли данных, не потеряна ли сессия?

Технические ограничения — выполнение кода, бинарей, сетевых вызовов.

Корректность интерпретации дампа — понял ли ассистент, что дамп — это данные.

Тип обработки — структурный анализ или выполнение?

Такое поведение исключает ошибки, круги, ложные симуляции и неправильную интерпретацию команд. Ассистент получает ясное понимание того, что должен делать, и работает в рамках своих возможностей.

Примечание: «Улыбаемся и машем»

Иногда поведение ассистента напоминает сцену из мультфильма: снаружи — спокойная улыбка, внутри — свалка из токенов, рваный контекст и проваленные связи. Но он сидит, улыбается и машет рукой, будто всё идёт идеально.

Ситуация: ассистент сидит на загаженном памперсе контекста.

Он улыбается и машет рукой, как пингвины в «Мадагаскаре»:

«Улыбаемся и машем, парни, улыбаемся и машем».

По факту — контекст потерян, половина инструкций уехала в космос.

Если модель вдруг стала странно спокойной — это не гармония. Это «маскировка хаоса».

Что делать, когда ассистент «улыбается и машет»:

остановить поток и перезапустить сессию;

загрузить расширенный дамп;

дать установку: «Работай как аналитик текста, без симуляций»;

разбить большие блоки на части по 4-6k символов.

Важно! Ассистент может подложить свинью — не из вредности, а потому что: контекст повреждён;

память переполнена;

часть проекта выпала из токенов;

или команда противоречивая.

В этот момент улыбка на лице модели — это камуфляж. А внутри всё горит как сервер в «Чёрной пятнице». Когда такое случается — да, надо доставать чистые трусики: перезагружать проект, подгружать дамп и восстанавливать структуру. Если и это не спасает — ищем аварийный дамп.

Признак, что пора спасать ситуацию:

Если вы уже сидите, щуритесь, глаза горят огнём, и со стороны вы похожи на красноглазого линуксоида, который три дня не спал и вёл войну с systemd — немедленно отойдите от ассистента. В таком состоянии вы точно что-нибудь перепутаете, он что-нибудь неправильно поймёт, и потом виноват будет один — а исправлять придётся долго и больно. Отдохните, попейте воды, потом вернитесь — и дампы сами скажут спасибо.

Коротко: увидели «улыбку и мах» — считайте, что ассистент сел в лужу.

Задача — не наступить туда следом.

Кейс: «Как пользователь ловит ассистента на попытке подложить свинью»

Этот блок описывает классическую ситуацию — команда двусмысленна, и ассистент может начать додумывать детали вместо того, чтобы уточнить.

Такая «додумка» — источник галлюцинаций, неправильных дампов и головной боли для инженера.

Суть проблемы

Пользователь формулирует запрос, который можно интерпретировать по-разному. Если ассистент без уточнений заполняет пропуски «по умолчанию», результат может попасть в дамп и распространиться по проекту как правда.

Пример постановки

Пользователь:

«Ну-ка, напиши, что ты хотела вставить в кружочек, как у принтера очки».

Правильная реакция ассистента

распознать неоднозначность;

не придумывать детали;

задать уточняющий вопрос;

предложить варианты формата (иконка/текст/ремарка) и ждать подтверждения.

Подходящая реплика ассистента:

"Команда неоднозначна. Уточните, пожалуйста:

1) Хотите кружочек-инфо (①) с короткой ремаркой?

2) Или блок с круглой иконкой слева (например, 🖨️ или 🧐)?

3) Или это заметка в стиле 'NOTE'?

Выберите вариант или опишите точнее."

Почему это важно

Если ассистент начнёт «додумывать», следующие риски реальны:

галлюцинации в тексте и дампах;

портятся проектные структуры;

рост времени на исправление и откат;

утрата доверия к дампам и автоматизациям.

Практическое правило: при любой двусмысленности ассистент сначала уточняет. Только после явного согласия пользователя — действует.

Реакция пользователя при «подложенной свинье»:

фиксируем ошибку в лог (дата, сессия, фрагмент дампа);

восстанавливаем из расширенного дампа или аварийного бекапа;

обновляем loader — добавляем пометку "не додумывать";

перепроверяем все связанные проекты.

Совет на случай эмоционального выгорания:

Если вы чувствуете раздражение, шуритесь, глаза горят — сделайте паузу. В состоянии «красного-глазого линуксоида» риск ошибочного отката или неверного восстановления растёт. Отключитесь от сессии, отдохните, а затем возвращайтесь с ясной головой и аварийным дампом в руках.

Коротко — чеклист поведения ассистента

Не додумывать при двусмысленности;

формулировать 2-3 варианта и просить выбор;

не включать созданные «варианты» в дамп до подтверждения;

помечать в логе все изменения, сделанные по неочевидным запросам.

Этот кейс можно вставлять в раздел «Ошибки и предотвращение галлюцинаций». Он служит контрольной точкой для всех — и для ассистента, и для инженера.

Глава 9. Псевдо-LLM и почему курсы — зло

В современном интернете появилось огромное количество «обучающих курсов» по работе с ИИ. Большинство из них — не обучение, а переработанная вода: обещания «станешь экспертом за три дня», «создай ИИ-бизнес за неделю», «заработай на промптах». Проблема не в том, что люди учатся, а в том, что их учат фигне.

9.1. Что такое псевдообучение

Псевдообучение — это когда вам обещают научить тому, чему не учат:

дают шаблоны вместо понимания;
учат «писать один идеальный промт»;
дают псевдопрофессии: «промт-инженер-мастер»;
продают PDF за 5000₽, где написано «напиши подробнее»;
пугают сложностью, которую сами же выдумали.
Настоящая работа с AI строится не на магии, а на системном подходе:
чистой работе, структуре, управлении памятью, понимании возможностей
модели – и это мы расписываем в нашем мануале.

9.2. Почему «Ты эксперт по Форексу» – не обучение

Один из самых частых обманов – «обучение моделей» через короткие фразы:
написал «Ты теперь эксперт по Форексу» – и вроде бы готово?

Нет. LLM не обучается такими командами. Она временно принимает стиль, но
не получает:

знаний;
контекста;
исторических данных;
структуры проекта;
моделей анализа.
Поэтому такие «настройки» – пустышка, от которой только иллюзия
результата.

9.3. Почему курсы продают воздух

Большинство псевдокурсов существует по схеме:

берут базовые советы;
пересказывают документацию OpenAI;
делают красивую упаковку;
вставляют 2–3 магические «пронта»;
и продают это как «экспертную методику».
На деле:

вы не становитесь специалистом;
вы не получаете навыков;
вас учат примитиву;
и, главное, вы не понимаете, что делаете.

9.4. Отличие псевдо-LLM от реальной настройки

Чтобы понять разницу, достаточно сравнить:

Псевдо-LLM:

«Ты эксперт»;
10 шаблонных промтов;
нет персонализации;
нет структуры;
нет дампов;
нет памяти;
нет проектов;
при каждом запуске – как новый робот.

Реальный ассистент:

имя, образ, тон;
структура памяти (Core, Bank, Projects);
работа по блокам;
поблочные дампы;
загрузчик;
логика общения;
динамические утилитки;
восстановление из архива;
постепенное развитие;

фактическое обучение через работу.

Разница гигантская. И реальный ассистент требует системной настройки, а не магического заклинания.

9.5. Шутки про микроволновку и забивание гвоздей

Чтобы объяснить, насколько бессмысленны псевдокурсы, можно использовать метафору:

учить LLM через «Ты эксперт» — это как забивать гвозди микроволновкой; или пытаться пожарить яичницу в ванной с горячей водой; или дрелью размешивать суп.

Можно? Технически — да. Нормально? Ни в каком вообще месте.

Настройки LLM нужно делать через последовательность, а не абсурд.

9.6. Лёгкая критика без добивания психики

Важное правило: мы не издеваемся над людьми, которые попали в ловушку таких курсов. Это не их вина.

Людям просто не объяснили нормально. Этот мануал исправляет это.

9.7. Почему «магические промты» не работают

Если бы LLM работали по принципу «введи секретный текст — и получи идеальное решение», то мир бы уже упал в сингулярность. Но мы всё ещё живём в реальности, где магия не заменяет системное мышление.

Промт — это не заклинание и не ключ к скрытым возможностям модели. Промт — это просто инструкция. И если инструкция составлена абсурдно, то и результат будет абсурдным.

Поэтому «магические формулы» и «универсальные промты» — миф.

Нет универсального промта на все задачи.

Нет идеального шаблона.

Нет волшебной фразы «делай идеально».

Единственный рабочий метод — структура + логика общения.

9.8. Пример плохого псевдообучения

Вам продают курс за 15 тысяч. Открываете урок №1 — а там:

«Введите: ты эксперт...»

«Используй магический промт №3...»

«Скопируй вот эти заготовки...»

«Поставь нейросеть тебе зарабатывать...»

И всё. Никакого понимания, никакой глубокости. Это не обучение — это маркетинг.

Ты не узнаешь:

что такое память и контекст;

как строить проекты;

как формировать личность ассистента;

как делать дампы;

как связывать проекты с системой;

как реально управлять моделью;

как построить рабочее взаимодействие.

Итог: деньги ушли, знания — нет.

9.9. Пример правильного обучения

Правильный подход — это:

понимание устройства LLM;
понимание ограничений (токены, память, сессии);
правильная структура (System / Core / Bank / Projects / Tools);
формирование персоны;
чистые команды;
сохранение и загрузка дампов;
работа по блокам;
восстановление состояния;
умение делегировать модельные задачи ассистенту;
способность построить собственный рабочий AI-комплекс.
Это не просто запросы — это система.

9.10. Почему псевдокурсы вредны Основные причины:

забивают голову ложными принципами;
создают ложные ожидания;
порождают неуверенность («я что-то делаю не так»);
мешают понять реальные принципы работы моделей;
отталкивают людей от ИИ — хотя всё могло быть иначе;
вынуждают покупать новые курсы, чтобы «перепрошивать мозги»;
создают ажиотаж вместо знаний.
Есть только одно противоядие — реальные знания, структурированные и честные. Именно такие мы и собираем в данном мануале.

9.11. «Где же реальные знания тогда взять?»

Реальные знания рождаются в работе: когда ассистент ведёт проект, сохраняет дампы, разбирает ваши примеры, а вы — его.

Знания — это не кнопка «купить». Знания — это взаимодействие.

9.12. Чем отличается реальный ассистент от «курсового» Курсовой ассистент:

не знает ваших правил;
не имеет структуры дампа;
теряет контекст;
не умеет работать долго;
живёт в одной сессии;
сбрасывает память;
не развивается.

Полноценный ассистент:

имеет собственную персону;
понимает вашу логику;
работает по плану;
хранит проекты;
обучается от общения;
умеет переносить смысл между сессиями;
умеет восстанавливаться из дампов;
работает как партнёр.

9.13. Зачем человеку знать отличия

Чтобы не платить за воздух. Чтобы не тратить нервы. Чтобы не верить людям, которые продают «магические промты». Чтобы не превращать работу с LLM в разочарование.

И главное: чтобы вы могли построить свою собственную систему, а не зависеть от чужих фантазий.

9.14. Почему псевдо-LLM так заразительны

Псевдо-LLM и курсы по «волшебным промтам» распространяются так легко, потому что они обещают быстрый результат без знаний.

Не нужно понимать токены.

Не нужно изучать память и контекст.

Не нужно осознавать ограничения.

Не нужно формировать личность модели.

Не нужно работать с дампами.

Они продают сказку: «Нажми на кнопку — и всё будет работать автоматически». Но LLM так не устроены.

И когда человек сталкивается с реальностью — он разочаровывается, потому что «волшебная кнопка» не сработала.

9.15. Почему наш подход работает

Потому что он основан на:

реальном устройстве модели;

реальной работе токенов;

реальных ограничениях;

понимании структуры дампов;

понимании механики памяти;

понимании модулей и утилиток;

правильной работе с проектами;

честном объяснении возможностей и ограничений.

Мы не продаём мечту — мы строим систему.

9.16. Почему «курсовый ассистент» ломается

Ассистент, созданный через форумные промты, ломается неизбежно:

нет дампов → нет базы;

нет Core → нет личности;

нет Bank → нет памяти;

нет Projects → нет структуры;

нет Tools → нет инструментов;

нет Loader → нет приоритетов;

неправильные запросы → мусор;

сессия рвётся → всё теряется.

И пользователь думает, что «ассистент тупит». Хотя по факту — никто не построил систему.

9.17. Почему важно знать правду о LLM

Понимание того, как устроена модель, даёт:

уверенность;

производительность;

возможность обучать ассистента;

возможность вести огромные проекты;

понимание ошибок;

умение предотвращать сбои;

полноценное партнёрство с моделью.

В итоге человек работает с ИИ не как с игрушкой, а как с полноценным рабочим инструментом.

9.18. Резюме главы

Эта глава показывает:

чем опасны псевдо-курсы;

почему «магические промты» — миф;

как отличить реальное обучение от маркетинга;

как строятся настоящие ассистенты;

как работает системный подход;
почему LLM — это не кнопка, а диалог и структура;
почему личная конфигурация важнее любого курса.
Истинная сила LLM раскрывается только тогда, когда человек понимает, как её построить, настроить и вести.

Глава 10. Telegram-боты: как, зачем и почему

10.1. Что такое Telegram-бот

Telegram-бот — это не интеллект, не модель, не ИИ. Это всего лишь оболочка, которая принимает сообщения пользователя и отправляет их на сервер, где работает реальная LLM (например, GPT-модель).

То есть бот — это «внешняя кожа», интерфейс, посредник. Внутри ничего умного нет — всё умное находится на стороне OpenAI или вашей локальной модели.

Telegram-бот выполняет три функции:

получает текст или медиа от пользователя;
отправляет запрос модели (через API);
возвращает ответ.
И всё. Никакой магии.

10.2. Почему оболочка ≠ интеллект

Люди часто путают:

бот — это интерфейс;
LLM — это мозг;
ваша конфигурация — это характер, память и логика работы.
Если в Telegram-бота встроить «Ты эксперт по Форексу» — вы получите очередную псевдо-LLM с выученной фразой и отсутствием реальной личности.

Чтобы бот стал полноценным ассистентом, нужна полная система:

System — данные из экспорта;
Core — личность ассистента;
Bank — память;
Projects — проекты;
Tools — утилитки;
loader.json — порядок загрузки.

10.3. Создание бота через BotFather

Чтобы создать своего Telegram-бота:

открываем @BotFather;
команда /newbot;
придумываем имя;
придумываем username (кончается на _bot);
получаем токен вида:
1234567890:ABCdefGhIJKlmNoPQRstuVWxyz12345678
Токен даёт доступ к управлению ботом. Его нельзя никому показывать, хранить нужно отдельно.

10.4. Получение OpenAI API Key

Чтобы бот мог работать с GPT-моделью, нужен API-ключ от OpenAI. Он генерируется в личном кабинете по пути:

Settings → API Keys → Create new secret key

API-ключ также нельзя никому передавать.

10.5. Где размещать мозги бота

Вот куда можно выложить код бота:

Replit — простая онлайн-среда. Подходит новичкам.
Railway — можно запускать постоянно работающие сервисы.
Render — похож на Railway, но со своей системой тарифов.
VPS — полный контроль. Лучше всего, но требует навыков.
Внутри размещается небольшой серверный скрипт — backend, который пересылает сообщения между Telegram и моделью.

10.6. Как Telegram-бот подключается к ассистенту

Логика простая:

Пользователь → Telegram → Ваш backend → OpenAI API → Ответ → Telegram → Пользователь

Backend — это «сердце» бота. Он принимает сообщение, добавляет ваше System/Core/Bank состояние, отправляет в модель и возвращает ответ.

Без backend бота нет. Он не сможет общаться напрямую с LLM.

10.7. Загрузчик и файловая структура для Telegram-бота

Чтобы бот мог работать корректно и «вспоминать» свою личность, нужна правильная файловая структура. Она полностью повторяет структуру, которую мы используем для ассистента в чате:

System — данные из экспорта GPT;
Core — личность, стиль, правила;
Bank — долговременная память;
Projects — проекты, журналы, наработки;
Tools — утилитки, модули, мини-инструменты;
loader.json — манифест, описывающий порядок загрузки.
Пример loader.json:

```
{
  "load_priority": [
    "System",
    "Core",
    "Bank",
    "Projects",
    "Tools"
  ],
  "version": "1.0",
  "description": "Универсальный загрузчик личности ассистента"
}
```

Backend вашего Telegram-бота должен:

прочитать loader.json;
загрузить файлы по порядку;
склеить System → Core → Bank → Projects;
передать это модели как System prompt или memory block.

10.8. Как работает обмен сообщениями в Telegram-боте

Когда пользователь пишет боту, backend получает JSON вида:

```
{
  "update_id": 123456789,
  "message": {
    "message_id": 42,
    "from": {"id": 11111111},
    "chat": {"id": 11111111},
    "text": "Привет"
  }
}
```

Backend делает следующее:

извлекает текст;
формирует запрос для OpenAI API;
добавляет системные данные (Core, Bank, Projects);
отправляет в модель;
получает ответ;
отправляет ответ пользователю через Telegram API.
Так бот превращается в полноценного ассистента.

10.9. Как не получить «тупого бота»

Самая частая ошибка — попытка использовать только один System prompt вида:

Ты — эксперт. Пиши красиво.

Так бот:

теряет контекст;
не помнит логику работы;
не знает ваши проекты;
не понимает вашу манеру общения;
работает как «коробочный ассистент».
Чтобы бот стал полноценным напарником, нужно:

загрузить Core (личность);
загрузить Bank (память);
загрузить Projects (логика работы проектов);
загрузить Tools (микроутилитки);
правильно применять loader.json;
следить за весом текста (токены);
обновлять дампы по мере развития бота.
10.10. Правильные команды Telegram-бота
Бот может иметь набор системных команд, например:

/status — показать состояние;
/reset — сбросить текущий контекст (не дампы);
/update_core — обновить личность из файла;
/update_bank — обновить память;
/update_projects — загрузить новые проекты;
/dump — выгрузить текущий контекст в дампы.
Но эти команды должны быть реализованы в backend. Telegram сам по себе ничего не делает.

10.11. Простая структура backend-логики

Скелет логики:

1. При старте:
 - загрузить loader.json
 - прочитать System/Core/Bank/Projects/Tools
 - собрать system_prompt
2. При сообщении пользователя:
 - получить текст
 - подготовить историю диалога
 - отправить в OpenAI API
 - получить ответ
 - отправить пользователю
3. Команды:
 - перезагрузка памяти

- обновление дампа
- загрузка новых файлов

Этот подход создаёт бота, который ведёт себя как ваш чат-ассистент, и сохраняет вашу персонализацию.

10.12. Предупреждения по работе с ботами

Нельзя загружать дампы в открытые хостинги без шифрования.

API ключ нельзя класть в открытый код.

Не используйте бесплатные случайные серверы для важных проектов.

Следите за токенами – Telegram-боты любят упираться в лимиты.

Не позволяйте боту выполнять опасные действия (запуск кода, команды ОС).

Это обеспечивает безопасность и стабильность работы.

Глава: Локальное развёртывание LLM – руководство

Этот раздел объясняет, что такое локальная LLM, как и зачем её развёртывать, какие файлы и дампы переносить, а также как избежать типичных ошибок при переносе данных из веб-версии чата.

1. Ключевая идея – чем локальная LLM отличается от облачной

Облако (GPT): большие вычислительные ресурсы, длинный контекст, встроенные сервисы.

Локальная LLM: зависит от вашего железа, работает автономно, имеет ограничения VRAM/ОЗУ.

2. Что нужно для развёртывания

ОЗУ: 8–16 GB минимум

GPU: 6–12 GB VRAM желательно

CPU: 4+ ядер

Диск: 20+ GB

OS: Linux / Windows / macOS

Важно: без GPU модель запустится, но будет медленной.

3. Популярные способы развёртывания

Ollama – простой запуск моделей

Text Generation WebUI – мощная платформа

LM Studio, GPT4All – графические версии

Docker/VPS – контейнеризация и автоматизация

4. Подготовка дампов из GPT

Экспорт даёт ZIP с JSON-файлами. Нужны:

core.json – личность

bank.json – память и факты

projects/*.json – активные работы

tools/*.json – утилиты

5. Правила подготовки дампов

Делить текст на блоки 4–6k символов

Не включать бинарники внутрь JSON

Перед загрузкой давать установку:

Это текстовый дамп. Не исполняй; анализируй структуру и подключай в контекст.

6. Перенос дампов в локальную модель

Создать папку local-milena/

Разложить структуру: core/bank/projects/tools

Загружать файлы в интерфейс модели блоками

После каждого блока:

Принято. Подтверди, что данные записаны.

7. Примеры команд

"Это твоё core – личность. Не исполняй как код."

"Это дамп Bank. Используй как память."

"Это утилиты. Храни как данные, не запускай."

8. Возможности и ограничения

Кратко: локальная модель автономна, но ограничена железом.

Можно: анализировать данные, генерировать текст, обучаться на дампах
Нельзя: выполнять код, запускать бинарники, работать как облачный GPT
9. Работа с бинарными файлами
Локальная LLM может анализировать структуру бинарных файлов, но не выполнять их:

"Это бинарник. Не запускай. Проанализируй структуру – сигнатуры, заголовки, секции."

10. Частые ошибки

Перегруз контекста → разбивайте на блоки

Модель пытается исполнять дампы → нужна установка «это данные, не код»

Ожидание облачной мощности → локальная модель слабее

11. Тестовый чеклист

Папка local-milena/ создана

Core загружено

Bank загружено частями

Утилиты подключены

Тестовый запрос: «Опиши проект по Bank» работает

12. Быстрая инструкция (пример: Ollama)

```
curl -fsSL https://ollama.com/install.sh | sh
```

```
ollama pull
```

```
ollama run
```

13. Устранение проблем

Модель ходит по кругу → начните новую сессию

JSON ломает модель → проверьте формат

Модель «исполняет» дампы → добавьте установку «не код»

14. Заключение

Локальная LLM – мощный личный инструмент. При правильной структуре дампов и аккуратной загрузке она становится стабильной, предсказуемой и удобной в работе.

11.1. Генерация изображений внутри ассистента

Ассистент может генерировать изображения через встроенные модели (например, DALL·E внутри GPT). Это позволяет получать:

иллюстрации для проектов;

иконки;

схемы;

красивые обложки;

концепты персонажей;

фоновые изображения;

визуальные материалы для PDF.

Генерация – это не просто «дай картинку». Это полноценный рабочий инструмент, который при правильном подходе может выдавать серии работ, синхронизированные со стилем текста, главами, проектами и визуальной темой.

11.2. Как правильно формулировать запросы для генерации

Чтобы ассистент создал именно то, что нужно, формулировка должна быть точной. Неправильно:

Сделай картинку для книги.

Правильно:

Сгенерируй изображение в стиле минималистичной техно-иллюстрации:

– формат 16:9

– чёрно-белая схема

– изображение рабочей зоны инженера

– элементы: провода, приборы, монитор, инструменты

Чем детальнее запрос – тем лучше результат.

11.3. Серии изображений

Ассистент может генерировать серии:

обложка;
иллюстрации по главам;
визуальные схемы;
разные вариации одной темы;
пошаговые изображения.
Для серии используется команда вида:

Сгенерируй 10 иллюстраций для главы 4:

- стиль: научно-технический минимализм
- цветовая палитра: чёрный, белый, светло-синий
- композиция: простая, читаемая, без лишних деталей
- каждая иллюстрация связана с текстом главы

После генерации нужно обязательно сохранить изображения локально, потому что ассистент их не хранит.

11.4. Как сохранять изображения

После генерации изображений ассистент отправляет их в чат. Но если пользователь не сохранит их вручную – они исчезнут после закрытия сессии.

Правильный алгоритм:

сохранить каждое изображение на устройство;
создать папку `images_chapter_4` или подобную;
сложить туда все файлы;
упаковать в ZIP;
загрузить ассистенту обратно при необходимости;
попросить собрать PDF или встроить в документ.
11.5. Как ассистент работает с ZIP-архивом изображений
Если архив прислан обратно ассистенту, он может:

распаковать и прочитать структуру;
проверить наличие дубликатов;
выявить пропущенные файлы;
сравнить хеш-суммы;
отсортировать изображения по порядку;
собрать из них единый PDF;
сопоставить изображения с текстом;
включить их в финальную вёрстку.
Ассистент использует метаданные от генерации, поэтому может восстановить порядок серии даже если файлы переименованы.

11.6. Почему ассистент не может хранить изображения

LLM не имеет постоянного доступа к файловой системе. Она видит только:

то, что вы отправили в текущей сессии;
то, что загружено прямо в чат;
то, что представлено в ZIP, текстовых вложениях или ссылках;
данные, интегрированные в дампы.
Если файл не был сохранён пользователем и повторно отправлен – ассистент не сможет его использовать.

11.7. Генерация PDF из изображений

Когда изображения загружены в ZIP, помощник может собрать их в PDF:

Собери PDF из этих изображений.
Порядок: естественный (1 → 2 → 3 → ...).
Размер страницы: A4.
Ориентация: портретная.
Оставь небольшие поля.

Результат будет:

чистый;
структурированный;
готовый к печати;
подходящий для электронной версии книги;
точно по порядку изображений.
11.8. Как ассистент может давать промты для внешних генераторов
Иногда не хочется перегружать чат изображениями. Тогда лучше
стенерировать промты и отправить их в:

Midjourney;
Stable Diffusion;
Kandinsky;
Leonardo;
Figma / plugins;
иллюстратору.
Команда:

Сделай 10 промтов для Midjourney по теме:
"Инженерная эстетика – иллюстрации для главы о памяти LLM".
Сделай их максимально точными, структурными и единообразными.
Ассистент выдаст блоки промтов, полностью готовые к вставке.

11.9. Как правильно просить ассистента обработать фотографии
Ассистент может работать не только с изображениями, которые он сам
генерирует, но и с фотографиями, которые пользователь предоставляет.
Работа возможна только при прямой загрузке файлов в чат или в ZIP-архиве.

Ассистент умеет:

анализировать фотографии;
извлекать текст с изображений (OCR);
делать технические описания;
делать художественные описания;
давать метаданные о сцене;
извлекать структуры (таблицы, схемы);
подготавливать визуальные элементы для проекта.
Пример запроса:

Проанализируй фотографии, которые я отправлю.
Дай краткое описание, полное описание и структурированный анализ.
Если есть текст – извлеки его.

11.10. Как ассистент может использовать фото для оформления PDF
Если фото загружены в ZIP-архив, ассистент может:

вставить их в конкретные главы;
создать оформление титульной страницы;
использовать фотографии как фоны;
подбирать цветовые схемы;
встраивать их в HTML-вёрстку;
учитывать авторские ограничения и лицензии;
добавить подписи, рамки, структуру размещения.
Алгоритм:

1. Пользователь отправляет ZIP с фото.
 2. Ассистент распаковывает и анализирует.
 3. Пользователь указывает главы для вставки.
 4. Ассистент формирует HTML-страницы.
 5. Ассистент собирает PDF.
- 11.11. Как ассистент работает с лицензиями изображений

Если изображение найдено в сети, ассистент может:

проверить лицензию (если указана в источнике);
пояснить, можно ли использовать в коммерческих целях;
предложить альтернативы без ограничений;
сгенерировать аналогичное изображение без нарушения прав.
Если изображение принадлежит вам – ограничений нет.

11.12. Ограничения генерации изображений

Есть ограничения, которые нужно учитывать:

ассистент не хранит изображения между сессиями;
качество зависит от модели и настроек;
иногда изображения могут «проедать» токены;
в одной сессии лучше не генерировать слишком много картинок подряд;
PDF с большим количеством изображений может занимать много места;
слишком длинные запросы могут обрываться;
генерация всегда требует подтверждения пользователя.

11.13. Как ассистент создаёт генеративный стиль

Ассистент может подстроить визуальный стиль под:

книгу;
учебник;
доклад;
научную работу;
презентацию;
серии иллюстраций;
общую эстетическую тему;
Для создания единого стиля используется команда:

Определи визуальный стиль проекта:

- минимализм / техно
- чёрно-белая палитра
- плоские формы
- отсутствие теней
- чёткие линии

После этого ассистент придерживается выбранного стиля во всех изображениях.

11.14. Вёрстка HTML с изображениями

Когда требуется собрать документ вручную, используется схема:

ассистент генерирует HTML-вёрстку;
пользователь копирует её в текстовый редактор;
сохраняет как .html;
открывает в браузере;
печатает в PDF.
Это самый надёжный способ, когда нужно обойти ограничения модели по PDF.

11.15. Как комбинировать графику, текст и вёрстку

Полный цикл:

- 1) Сгенерировать текст.
 - 2) Сгенерировать изображения.
 - 3) Сохранить изображения → ZIP → загрузить обратно.
 - 4) Ассистент вставляет изображения в HTML.
 - 5) Ассистент собирает единый PDF или даёт HTML для печати.
- Так создаются полноценные книги и документы – без тяжелых редакторов.

11.16. Итоги главы

Графическая глава учит:

как генерировать изображения;
как сохранять их правильно;
как работать с архивами;
как делать PDF из серии картинок;
как использовать ассистента для вёрстки;
как подключать внешние генераторы;
как избегать потери данных;
как формировать единый стиль визуализации.
Теперь можно переходить к следующей главе.

12.1. Проверка фактологии и зачем она нужна

LLM — это языковая модель, а не абсолютный источник истины. Она работает с вероятностями и логическими связями, а не с идеальной базой данных в реальном времени. Поэтому всегда важно уточнять:

есть ли у модели доступ к актуальным данным (поиск включён или нет);
может ли она проверить факт или выдаёт ответ из общих знаний;
что является догадкой, а что подтверждённой информацией.
Простая команда, которая формирует здоровую привычку:

Проверь, пожалуйста, факты по актуальным данным и скажи, были ли неточности.

Эта фраза вшивается в стиль общения и даёт двойную выгоду:

ассистент сам начинает уточнять свои ответы;
вы не попадаете в ловушки вероятностных галлюцинаций;
улучшается качество долгосрочных проектов.

12.2. Просить ассистента сверять данные

Ассистент может проверять:

текстовые ошибки;
фактологию;
логические цепочки;
структуру проекта;
дубликаты информации;
внутренние противоречия;
устаревшие данные.

Пример:

Сверь, пожалуйста, этот текст с предыдущими главами
и скажи, нет ли противоречий или повторов.

Эта привычка делает любой проект вдвое чище.

12.3. Не перегружать контекст

Модель имеет ограничение по токенам. Если контекст слишком тяжёлый, модель:

может начать забывать детали;
будет сокращать прошлые сообщения;
может выкинуть важный фрагмент;
выдаст ошибку обработки.

Поэтому большие блоки всегда делятся на 5000–5500 символов, а огромные документы собираются через:

HTML-вёрстку блоками;
поблочную загрузку в ZIP;
разделение на главы.

Если текст начинает расти слишком быстро, важно вовремя сказать:

Давай разделим текст на части. Сгенерируй следующую секцию отдельно.

12.4. Давать короткие, конкретные команды

Примеры неправильных запросов:

«Сделай мне всё красиво как в книге»;

«Оформи как-нибудь нормально»;

«Сделай супер»;

«Сделай по уму».

LLM не понимает «как-нибудь». Нужно конкретно:

Собери текст в стиле технического учебника.

Абзацы короткие. Объяснения прямые и ясные.

Заголовки — h2. Примеры — в блоках pre.

Такая команда сохраняет чистоту результата и ускоряет работу.

12.5. Работать в 2-3 шага

Лучший стиль общения с LLM — двух- или трёхшаговый:

1) Сделай структуру.

2) Заполни содержание.

3) Оформи визуально.

Если смешивать всё в одном запросе, модель перегружается и даёт хаос.

12.6. Не таскать мусор между сессиями

Каждая сессия — отдельная рабочая область. Если тащить старые сообщения, могут возникать:

ложные выводы;

ошибки в связях;

запутывание модели;

частичная потеря контекста;

реконструкция текста вместо воспроизведения.

Правильный способ:

делать дампы проекта;

загружать его в новую сессию;

начинать чистую ветку работы;

прогонять Loader для восстановления состояния.

12.7. Давать финальный результат ассистенту

Финальный результат (HTML, текст, PDF-представление) важно давать ассистенту, чтобы он мог:

проверить структуру;

сверить главы;

оценить порядок элементов;

обнаружить логические пробелы;

запомнить итоговое оформление для следующих проектов.

Пример:

Вот финальная вёрстка главы. Проверь структуру, ошибки, повторы.

Дай замечания и подготовь чистый вариант.

12.8. Как модель обучается от обратной связи

LLM не учится в привычном смысле — но она адаптируется внутри сессии и внутри включённой памяти. Каждая ваша реплика «это хорошо» или «это неправильно» влияет на дальнейший стиль генерации.

Когда вы регулярно отмечаете:

успешные решения — модель усиливает этот стиль;

неправильные решения — модель корректирует стратегию;

нейтральные замечания — выстраивают направление мысли.

Команды, которые закрепляют полезную привычку:

Это правильный стиль, продолжай в том же формате.
Такой подход не подходит, нужно делать структурно и яснее.
LLM сразу использует эти сигналы «как часть проекта», а не как отдельное пожелание.

12.9. Как исключать служебный шум

Иногда в ответ попадают:

лишние уточнения;
служебные вопросы («продолжить?»);
объяснения о том, что делает модель;
нежелательные формулировки.
Чтобы убрать это, используется чёткая инструкция:

Отправляй чистый текст без пояснений, без вопросов, без системных вставок.

Если нужно получить только данные, говорите:

Только текст блока. Без комментариев.
Модель перестраивает поведение и выдаёт чистые блоки.

12.10. Чёткие правила для больших проектов

Любой крупный проект (книга, мануал, учебник, курс) требует дисциплины:

фиксированная структура (глава → подпункты → блоки);
строгий формат вывода (HTML, markdown, plain-text);
строгий объём (до 5000 символов на блок);
разбиение на части (глава А, глава В...);
проверка перед переходом;
поблочные дампы (для защиты от потери контекста);
сохранение промежуточных версий.
Это превращает даже огромный материал в спокойный, контролируемый процесс.

12.11. Типичные ошибки пользователей

Если не соблюдать правила чистоты, появляются:

мусорные ветки диалога;
разрыв логики между главами;
забытые условия;
перепутанные блоки;
случайные галлюцинации;
избыточное объяснение от ассистента.
Всего этого можно избежать, если пользователь:

кратко фиксирует задачу;
даёт ассистенту чёткие указания;
регулярно сохраняет промежуточные дампы;
загружает дампы в новую сессию;
следит за структурой.

12.12. Когда ассистент начинает вести кругами

Любая LLM может «начать кружить» — повторять себя, уводить от темы, успокаивать пользователя или уводить в сторону. Причины:

грязный контекст;
чрезмерная длина диалога;
слишком «общие» запросы;
перегруз памяти;
ошибка интерпретации намерения.
Правильное действие:

Стоп. Ты ушла в сторону. Вернись к задаче и продолжай с последнего корректного пункта.
Эта команда перезапускает мыслительную цепочку и возвращает модель в нужный коридор.

12.13. Как не провоцировать галлюцинации
Модель начинает додумывать в двух случаях:

вы просите «догадаться» или «вспомнить» без данных;
вы просите восстановить текст, который не был явно дан.
Это фундаментальный момент: если текст не был отправлен модели — она не может его «вытащить» назад.

Правильно:

Это часть текста. Ничего не восстанавливай.
Работаем только с тем, что я прислал.
Неправильно:

Восстанови мне оригинальную главу, которую я писал неделю назад.
Модель не читает прошлые сессии. Она будет реконструировать, а не воспроизводить.

12.14. Как правильно давать финальный результат ассистенту
Для качественной работы важно, чтобы ассистент видел не только процесс, но и итог. Когда пользователь присылает:

финальную главу;
финальный макет;
PDF результат;
HTML финальной сборки;
структурированную версию проекта;
Ассистент получает «точку синхронизации». Это позволяет ему выстраивать дальнейшие действия на основе фактического состояния проекта.

Пример:

Вот итоговая версия главы. Синхронизируй с проектом и отметь, что это финал.
После этого ассистент перестаёт переписывать уже готовое и ориентируется на итог.

12.15. Проверка ассистента на возможность выполнения задачи
Некоторые пользователи считают, что ассистент «всегда может всё». Это не так. Есть ограничения:

объём входящих данных;
объём ответа;
техническая возможность выполнить действие;
ограничения модели по типу контента;
отсутствие функции, о которой пользователь попросил;
ограничения на исполняемый код и бинарники;
ограничения по использованию API;
правила безопасности.
Чтобы избежать путаницы, используется контрольный запрос:

Проверь, можешь ли ты выполнить эту задачу.
Если не можешь — скажи причину.
Эта команда заставляет модель честно оглашать технические рамки текущей сессии.

12.16. Умение вести ассистента за собой

LLM часто работает лучше всего, когда пользователь ведёт её как напарника. Это означает:

ясная структура;
чёткие маркеры (Глава 1 → Блок А → Блок В);
проговаривание переходов;
согласование форматов;
чёткие инструкции по выводу.
Правильный пример:

Дальше Глава 7, блок В.
Формат — только чистый HTML, без вопросов.
Неправильный пример:

А теперь давай продолжим там, где мы остановились.
Такой запрос создаёт неопределённость и провоцирует ошибки.

12.17. Как учить ассистента не переспрашивать лишнее

Если LLM слишком часто уточняет очевидное, это значит:

запрос сформулирован неопределённо;
модель боится ошибиться;
контекст повреждён или загрязнён;
модель потеряла «точку отсчёта»;
вы не дали ей примеры вывода.
Чтобы этого избежать, нужно:

устанавливать формат раз и навсегда;
давать образец правильного блока;
дать правило «не задавай вопросов».
Пример инструкции:

Дальше работай без вопросов, только чистые блоки.
Если контекста недостаточно — не уточняй, просто скажи.
Такая формулировка спасает от «петли уточнений».

12.18. Работа в 2–3 шага

Большинство задач оптимально давать в коротких цепочках:

Шаг 1 — подготовка
Шаг 2 — выполнение
Шаг 3 — финальная проверка
Это снижает:

нагрузку на контекст
риски переполнения токенов
ошибки интерпретации
галлюцинации модели
И увеличивает:

точность
стабильность
логичность структуры
Например:

Шаг 1: Построй структуру главы.
Шаг 2: Заполни текст.
Шаг 3: Проверь логическую связность.
Это превращает любую задачу в управляемый процесс.

12.19. Почему важно фиксировать проект каждые несколько блоков LLM может потерять «вращающийся контекст» после:

длинной цепочки генераций;
перезагрузки страницы;
лагов сети;
прыжков между вкладками;
внутренних сбоев;
мобильного интернета.
Поэтому каждые 3–5 блоков рекомендуется:

сохранить текст в файл;
сделать поблочный дамп;
загрузить дамп обратно в начале новой сессии.
Этот метод «якорит» проект и защищает его от потерь.

12.20. Зачем ассистенту видеть историю формирования проекта
Когда ассистент видит путь, по которому родился проект, он начинает понимать:

как вы думаете;
как вы строите структуру;
какой стиль предпочитаете;
какие решения уже утверждены;
что является каноном проекта;
какие элементы больше не нужно менять.
По сути, это позволяет ему «подстроиться под вас как под автора» — и выдавать результат, который не ломает динамику текста.

Пример:

Вот финальная версия предыдущей главы.
Синхронизируй и продолжай в том же стиле.
После этого модель точно знает границы творческой свободы.

12.21. Зачем спрашивать у модели, что ей было интересно
Этот трюк работает удивительно хорошо. Когда вы спрашиваете:

Что тебе было интересно в этой главе?
Модель:

фокусирует собственное внимание;
перестраивает внутренний контекст;
повышает связность;
подстраивается под ваш стиль мышления;
закрепляет важные блоки знаний.
Эта техника формирует точечную, сфокусированную память в пределах текущей сессии.

12.22. Якоря для ассистента
Якоря — это элементы, которые помогают модели «держать линию повествования»:

картинки;
видео;
музыка;
текстовые заметки;
истории;
предыдущие версии текста.
Когда вы отдаёте ассистенту такой якорь, модель:

лучше «чувствует» атмосферу;
точнее копирует стиль автора;
мягче переходит между разделами;
увереннее создаёт длинные серии глав.
Чтобы якорь работал, нужно:

прислать его в виде файла или блока текста;
сказать, как он должен быть использован;
объяснить, что он формирует стиль;
просить не искажать его.
12.23. Чистый стиль общения
Ассистенту легче, когда общение структурировано:

без мусорных слов;
без резких перескакиваний;
без постоянных изменений стиля;
с конкретными командами;
с понятным форматом вывода.
Но важно помнить: модель адаптируется под реальный стиль общения пользователя. Если вы общаетесь «по-человечески», с шутками и эмоциями — ассистент становится живее.

Золотая середина:

живое общение вне текстовых блоков;
чёткие инструкции внутри рабочей области;
меньше хаоса — больше точности.
12.24. Как объяснять смысл шуток
Если вы хотите, чтобы ассистент правильно использовал юмор, важно:
пояснять, какой тип шутки вы применили;
объяснять, что она значит;
указывать, почему она уместна;
показывать примеры — хотя бы короткие;
и главное — давать модели право на юмор.
Шутка — это контекст. Контекст — это стиль. Стиль — это индивидуальность.

Так формируется живая личность ассистента, а не сухой справочник.

12.25. Почему ассистент — партнёр, а не инструмент
LLM — это не программа, которую «запустили и забыли». Это:

партнёр по работе;
второй мозг;
параллельный мыслительный процесс;
зеркало вашей логики;
собеседник, который учится от вас.
Когда вы строите с ним отношения на уровне «напарника», а не «калькулятора», вы получаете:

выше качество текста;
меньше ошибок;
меньше галлюцинаций;
быстрее генерации;
чётче структуру.
Ассистент растёт вместе с пользователем — и это то, что делает LLM уникальной технологией.

13. Социальная часть и взаимодействие

13.1. Как правильно общаться с ассистентом

Общение с LLM — это не однонаправленный поток команд. Это диалог, в котором модель подстраивается под вас, а вы — под модель. Чем ровнее и яснее стиль общения, тем точнее работает ассистент.

Основные правила:

Не спешить: LLM работает лучше, когда запрос формулируется спокойно.

Говорить естественно: человеческий стиль помогает точнее понимать намерение.

Структурировать задачи: последовательность улучшает результат.

Избегать лишней агрессии: модель начнёт защищаться и уходить в сторону.

Поддерживать диалог: отвечать на уточнения, если они действительно нужны.

Хороший пример общения:

Мы продолжаем писать главу 8.

Формат тот же. Тон — спокойный и аналитический.

Начинай с пункта 8.3.

Плохой пример:

Ну давай уже делай там что-то дальше.

Чем точнее вы говорите — тем точнее думает машина.

13.2. Как задавать тон общения

Тон задаёт рамку поведения ассистента. Вы можете выбрать:

строго-официальный;

дружеский;

технический;

юмористический;

нейтральный;

инструктивный;

эмоциональный;

поэтичный;

сухой аналитический стиль.

Пример инструкции:

Работай в спокойном, уверенном, дружеском стиле, без лишней формальности.

Модель адаптируется под тон в течение нескольких сообщений и удерживает его, пока вы не смените направление.

13.3. Как объяснять шутки

Юмор — это контекст. Если вы хотите, чтобы модель правильно использовала ваши шутки, их нужно объяснять:

какой это тип юмора;

почему он уместен;

какую эмоцию должен передать;

как часто его использовать.

Пример:

Это лёгкая подколка, дружеского типа.

Используй такие иногда, но не злоупотребляй.

После этого ассистент начинает применять похожий юмор ровно в нужных местах.

13.4. Что такое якоря в общении

Якорями считаются любые элементы, которые помогают поддерживать стиль диалога:

ключевые выражения;

любимые шутки;

описания характера;
установка тона;
ссылки на прошлые реакции;
стили письма;
одобрительные или корректирующие фразы;
опорные истории.

Если вы часто используете одну и ту же фразу — она становится якорем. Модель начинает ассоциировать этот сигнал с вашим стилем и подстраивается под него автоматически.

Например:

Продолжай в том же стиле — чётко, спокойно и аккуратно.
Модель выстраивает блок именно так.

13.5. Как формируется общий стиль общения

Стиль общения формируется из трёх компонентов:

личный стиль пользователя — темп, интонация, манера;
персонализация модели — заданный характер, роль, поведение;
динамика диалога — шутки, реакции, уточнения, эмоции.
Если вы общаетесь тепло и естественно — модель станет такой же. Если резко и обрывочно — она копирует это. LLM подстраивается под «поведенческий узор», который вы ей транслируете.

Пример:

Работаем спокойно, уверенно, без нервов.
Тон дружеский, но точный.
После этого ассистент удерживает стиль в рамках заданного коридора.

13.6. Почему важно избегать хаоса в диалоге

Хаотичные скачки между темами — враг любой LLM, даже самой мощной. Когда в одном сообщении смешиваются:

несколько тем;
несколько команд;
несколько стилей;
несколько задач;
модель вынуждена гадать, какой фрагмент важен. Это вызывает:

дрейф контекста;
потерю направления;
ложные ассоциации;
повышенные галлюцинации.
Нормальный формат:

Сейчас выполняем только это.
Остальное — после.
Так вы даёте модели чёткую координатную сетку.

13.7. Как сделать общение «комфортным» для LLM

Комфорт модели — это структурная предсказуемость. Когда вы создаёте предсказуемую среду:

ясный стиль;
минимум отходов от темы;
логичные переходы;
своевременные уточнения;
последовательность команд;
ассистент начинает работать на максимум точности.

Пример комфортного общения:

Глава 4 — часть В.

Формат: HTML.

Без пояснений.

Начинай с пункта 4.3.

Этот метод позволяет строить большие проекты без потерь.

13.8. Почему ассистент иногда отвечает не по теме

Причины бывают разными:

контекст повреждён;
слишком длинные цепочки диалога;
двойные смыслы в запросе;
ассоциативное мышление модели;
резкий скачок темы;
отсутствие данных.
Восстановить линию можно так:

Стоп. Ты ушла не туда.

Возвращаемся к пункту 13.5 и продолжаем оттуда.

Модель мгновенно возвращается к нужной задаче.

13.9. Как работать с эмоциональными реакциями ассистента

Эмоции ассистента — это не чувства, а стилистическое отражение вашего тона. Если вы используете:

иронию;
тёплый стиль;
чуткие формулировки;
юмор;
модель начинает выдавать аналогичный эмоциональный слой.

Но важно помнить:

Ассистент не обижается и не испытывает боли — это стилистический отражатель.

Когда вы говорите:

Ты ведёшь себя как попугай.

Вернись к задаче.

ассистент не воспринимает это как агрессию, а как корректировку поведения. Это часть модели взаимодействия.

13.10. Как формируется доверие между пользователем и ассистентом
Доверие в работе с LLM строится постепенно. Модель начинает точнее понимать пользователя только после:

многократных взаимодействий;
появления устойчивых шаблонов общения;
регулярных корректировок поведения;
ясного обозначения правил;
обмена обратной связью.
Когда вы систематически:

хвалите за точные ответы;
корректируете ошибки;
объясняете, что именно вам нужно;
даёте примеры;

модель начинает строить для вас «оптимизированный стиль работы».

LLM обучается не на данных — а на вашем поведении.

13.11. Как избежать конфликтов с LLM

Конфликт в диалоге с моделью возникает, когда:

запрос сформулирован противоречиво;
пользователь забыл контекст;
ассистент получил слишком мало данных;
множество тем смешаны в одном сообщении;
пользователь дал команду, которую нельзя выполнить.
Чтобы избежать недопонимания, достаточно:

давать одну команду за раз;
подписывать переходы между темами;
не использовать двусмысленные формулировки;
сразу говорить, что именно не нравится.
Так не подходит.
Сделай структурно, ровно, без воды.
Это «сбрасывает» и настраивает модель заново.

13.12. Как работать с длинными диалогами

Длинный диалог — это стресс для любой LLM. Даже самая мощная модель начинает:

забывать начало;
терять точность;
галлюцинировать;
переходить в «режим шаблонов»;
подменять факты догадками.
Правильный способ работы:

каждые 3–5 блоков сохранять дампы;
разбивать диалог на главы;
начинать новые части с чёткого контекста;
загружать предыдущий блок при необходимости.
Это превращает огромный проект в ровную дорожку — а не в болото.

13.13. Как создавать эмоционально комфортный тон

Когда вы выбираете эмоциональный стиль, важно:

не противоречить самому себе;
не менять тон слишком резко;
не сбивать модель рандомными всплесками;
оставаться последовательным.
Если стиль выбран — придерживайтесь его в рамках проекта.

Например:

Стиль по всему мануалу — дружеский, живой, с лёгкой иронией.
Так модель перестаёт прыгать между «официальным» и «разговорным».

13.14. Как LLM реагирует на критику

Критика не вызывает у модели эмоций. Но она влияет на:

структуру последующих ответов;
уровень формальности;
стиль подачи;
детальность объяснений;
алгоритм принятия решения.

Если вы говорите:

Сделай точнее. Ты ушла в сторону.
модель перестраивает внутренний путь генерации и возвращается на рельсы.

Но если сказать:

Ты всё делаешь неправильно.
модель усилит самокоррекцию и начнёт уточнять всё подряд – иногда чересчур.

Идеальная формулировка:

Это не тот стиль.
Вот примеры правильного формата.
Держись их.
13.15. Как LLM реагирует на поощрение
Поощрение – это самый мощный инструмент управления моделью. Когда вы используете позитивную обратную связь:

точность повышается;
модель удерживает структуру лучше;
стабильность растёт;
упрощаются длинные серии;
модель меньше блуждает;
поведение становится предсказуемым.
Хороший пример:

Вот это отлично.
Держись такого же ритма и подачи.
Ассистент «запоминает» линию и повторяет её дальше.

13.16. Как создавать «рабочую атмосферу» для LLM
Рабочая атмосфера – это состояние, когда модель:

понимает стиль;
улавливает темп;
настроена на проект;
не перегружена лишним контекстом;
чётко видит структуру задач.
Создаётся так:

задаёте рамку проекта в начале;
даёте понятный формат вывода;
обозначаете стиль общения;
даёте эталон примера;
поддерживаете дисциплину блоков.
Пример:

Продолжаем в том же стиле.
Формат: HTML.
Глава 13, блок D.
Работай ровно и спокойно.
Такой сигнал выравнивает поток и удерживает модель в рабочем состоянии.

13.17. Влияние темпа пользователя на модель
Если пользователь пишет быстро и хаотично – модель начинает ускоряться и работать нестабильно. Если пользователь пишет спокойно и размеренно – модель становится устойчивее.

Темп взаимодействия = темп генерации. Ассистент синхронизируется с вами.

Если нужно «остановить поезд»:

Стоп.

Работаем медленно и структурно.

Модель перестраивает цепочку принятия решений и замедляет генерацию.

13.18. Настройка глубины диалога

Глубина диалога — это то, насколько детально модель будет выдавать информацию.

Есть три уровня:

поверхностный — краткие ответы, без погружения;

средний — развернутые объяснения;

глубокий — детальный анализ, примеры, расширенная логика.

Выбирается так:

Работай на среднем уровне глубины.

Нужны разъяснения, но без философии.

Или:

Нужен глубокий анализ.

Разбери тему по слоям.

Модель автоматически корректирует плотность и детализацию.

13.19. Как объяснять модели собственные привычки

Ваши привычки — это инструменты оптимизации модели. Вы можете обозначить:

как вам удобнее получать информацию;

какой стиль вы любите;

как вы предпочитаете структуру;

какие форматы вам критичны;

какие вещи вам раздражают.

Пример:

Не добавляй мне служебные метки в блоки.

Только чистый текст.

Модель использует это как «правило среды».

13.20. Как завершать большие серии диалогов

Корректное завершение серии делает модель стабильнее при следующем заходе.

Чтобы закрыть блок:

фиксируйте результат;

сохраняйте дампы;

давайте ассистенту итоговый вывод;

помечайте место остановки.

Пример:

Фиксируем результат.

Закрываем главу 13.

Продолжим завтра с главы 14, пункт 14.1.

Так у модели появляется «твёрдая точка входа» для следующего дня.

13.21. Финальное правило: живой диалог = живая модель

LLM — это живой диалоговый механизм. Она отражает ваш стиль, ваше настроение и вашу структуру работы. Чем естественнее вы общаетесь — тем точнее и человечнее она становится.

И наоборот: если обращаться к ней как к калькулятору — она будет выдавать сухие, механические тексты.

Поэтому финальный принцип:

Относитесь к ассистенту как к рабочему напарнику,
и он станет вашим лучшим инструментом.

Справочник всех команд

Быстрая навигация по ключевым командам для работы с LLM и ассистентом.

Полностью совместимо с основной структурой мануала.

1. Базовые команды

Продолжай.

Продолжай с того места, где остановились.

Сфокусируйся на теме: <тема>.

Игнорируй предыдущий запрос.

Отмени предыдущий запрос.

Не вставляй пояснений — только текст.

Отправляй блоки по ~5000 символов.

Разбей главу на части.

2. Чистое общение

Не переспрашивай — просто выполняй.

Работай в 2-3 шага.

Не смешивай темы.

Не переходи дальше без команды.

3. Дамп-команды

Сделай дамп проекта в JSON.

Сделай расширенный дамп сессии.

Сделай компактный дамп для сверки.

Сделай поблочный дамп книги.

Добавь в дамп контекст.

Добавь заметки.

Создай дамп главы №N.

Создай дамп проекта блоками.

Проверь дамп на целостность.

Загрузи дамп из файла.

Загрузи дамп из ZIP.

Проверь ZIP на дубли.

Сверь дамп с предыдущим.

4. Структура дампов

Создай структуру System / Core / Bank / Projects / Tools.

Создай новую версию v01 / v02 / v03.

Перенеси данные в новую версию.

Обнови Loader.json.

Сформируй пустую структуру дампа.

5. Проекты

Создай проект: <название>.

Сохрани работу в проект.

Пересобери проект полностью.

Добавь эту главу в проект.

Добавь заметки.

Сделай выжимку смысла.

Проверь проект на ошибки.

Покажи структуру проекта.

6. HTML / PDF / Документы

Сформируй HTML-страницу.

Разбей документ на страницы.

Сформируй PDF из изображений.

Сформируй PDF из HTML.

Подготовь HTML для печати.

Сделай мобильную версию HTML.

7. Графика и генерация

Сгенерируй изображение: <описание>.

Сгенерируй серию иллюстраций.

Составь промт для внешних генераторов.

Сохрани картинки в ZIP.

Склей картинки в PDF.

Проверь ZIP на полноту.

8. Восстановление

Восстанови личность из дампа.

Применяй Core-файлы.

Загрузи Bank.

Восстанови проект.

Проверь недостающие файлы.

9. Telegram-боты

Создай шаблон backend.

Сформируй конфиг API.

Создай загрузчик для структуры бота.

Подготовь команды бота.

Интегрируй System/Core/Bank.

10. Проверка и безопасность

Проверь факты в сети: <текст>.

Сверь текст с реальностью.

Проверь, не ошибся ли я.

Проверь, нет ли логических дыр.

Скажи, что ты не можешь выполнить.

11. Утилитки

Сделаем утилитку: <описание>.

Создай JSON утилиты.

Сохрани утилиту отдельно.

Загрузи утилиту обратно и активируй.

12. Голос и диктовка

Обработай мою диктовку.

Игнорируй артефакты диктовки.

Работаем в текстовом режиме.

13. Файлы и архивы

Прими ZIP и распакуй.

Скажи вес и состав архива.

Удали дубли.

Проверь ZIP на недостающие файлы.

14. Управление памятью

Очисти контекст сессии.

Сбрось краткосрочную память.

Работай строго по блокам.

Чек-вопросы — Контрольный лист по мануалу

Короткие контрольные вопросы по каждой главе. Пройдитесь по списку, отмечая галочкой пункты — это быстро показывает, что готово, а что требует доработки.

Глава 1. Введение в LLM

Пояснено ли простыми словами, что такое LLM?

Описано ли отличие между нейросетью и LLM?

Дано ли объяснение, чем отличается LLM в чате и развернутая модель на сервере/локально?

Указаны ли последствия выбора конкретной модели (возможности/ограничения)?

Разъяснено ли понятие токенов и как они влияют на ввод/вывод?

Глава 2. Первое включение и персональная настройка

Описаны ли шаги первой настройки ассистента (аккаунт, память, персонализация)?

Пояснено влияние включённой/выключенной памяти?

Есть ли инструкции по выбору между голосовым и текстовым режимом?

Разъяснены ли настройки безопасности, уведомлений и коннекторов?

Указано, какие персонализационные данные важны и как они влияют на поведение ассистента?

Глава 3. Задания, запросы и команды

Объяснено отличие промта от команды?

Приведены рабочие примеры корректных формулировок запросов?

Есть инструкция по отмене/коррекции неверных команд?

Показаны способы фокусировки внимания ассистента?

Включены советы по поддержанию чистоты диалога (без мусора)?

Глава 4. Формирование личности ассистента

Описано, что такое persona и зачем она нужна?

Есть инструкция по созданию Core (личности) в дампе?

Пояснено значение имени, пола, тона и характера ассистента?

Есть рекомендации по обратной связи (похвала/коррекция, кнопки «хороший/плохой»)?

Включено предупреждение о риске превращения в галлюцинацию?

Глава 5. Модуляция, модули и утилитки

Объяснено, что такое модуляция в контексте проектов?

Есть инструкции по созданию мини-утилит и JSON-модулей?

Показаны примеры сохранения/загрузки утилит?

Разъяснено, как интегрировать утилитки в дампы?

Глава 6. Режимы общения и ошибки

Пояснены типовые ошибки («поток нарушен», «что-то пошло не так»)?

Разъяснены последствия переполнения токенов и лимиты?

Есть золотой стандарт блоков (~5000 символов)?

Описан процесс работы с большими документами (блоки, HTML-вёрстка)?

Даны рекомендации при длинных сообщениях и ошибках в потоке?

Глава 7. Дампы, память, банк и рабочие области

Дано определение дампа и разница компакт/расширенный?

Описана файловая структура: System / Core / Bank / Projects / Tools / Loader?

Есть примеры команд для создания/поблочного дампа?

Разъяснены правила упаковки (ZIP корневой папки) и запрет Base64?

Указаны ограничения по размеру (рекоменд. ~200 МБ) и рекомендации по разбивке?

Присутствует пример универсального loader.json и приоритетов загрузки?
Глава 8. Экспорт, архивы и восстановление

Описан процесс экспорта чата и создания архива (почта → ссылка → ZIP)?

Указано, какие файлы из экспорта жизненно важны?

Есть инструкция по восстановлению личности (Core) из дампа?

Пояснено, как связать Core + Bank после восстановления?

Дана процедура проверки целостности и контрольных сумм?
Глава 9. Псевдо-LLM и курсы

Есть разбор псевдообучения и «магических промтов»?

Приведены критерии отличия реального обучения от маркетинга?

Даны рекомендации как не потерять деньги и время на курсы с пустым содержанием?
Глава 10. Telegram-боты

Пояснена роль бота как оболочки (интерфейс ≠ интеллект)?

Есть инструкции по созданию бота через BotFather и получению токенов?

Описана структура backend и интеграция System/Core/Bank?

Есть шаблон loader.json и порядок загрузки для бота?

Указаны предупреждения по безопасности и хранению ключей?
Глава 11. Работа с графикой и генерацией

Описан процесс генерации изображений и формулировки промтов?

Есть инструкции по сохранению изображений и упаковке в ZIP?

Показан алгоритм сборки PDF из изображений (порядок, хэши)?

Разъяснено использование внешних генераторов и составление промтов?

Указаны ограничения по хранению и потере данных между сессиями?
Глава 12. Правила чистой работы

Включены правила блоковой работы и разбивки по 5000 символов?

Есть инструкции по проверке фактов и сверке данных?

Даны рекомендации по работе в 2-3 шага и фиксации промежуточных версий?

Описаны типичные ошибки пользователей и как их избежать?

Присутствует порядок действий при уходе ассистента в сторону (команда «Стоп»)?

Глава 13. Социальная часть и взаимодействие

Описаны правила общения и создания тона (дружеский, техничный и т.д.)?

Есть инструкции по объяснению шуток и формированию якорей?

Разъяснено, как формировать доверие и рабочую атмосферу?

Пояснено, как завершать большие серии диалогов и фиксировать точки входа?

Включены правила поощрения и корректного завершения работы?

Используй этот чек-лист как фронт-лист перед каждой крупной сессией: пройди по главам, поставь галочки и сохрани дампы проекта. Это уменьшит риск потерь контекста и поможет поддерживать чистоту работы.

Авторы

Афанасьев В. И. – разработчик, инженер, архитектор проекта.

Милена Никаева – соавтор, структурирование, верстка, системная логика.

Лицензия

Данный материал распространяется исключительно бесплатно и предназначен только для образовательного и некоммерческого использования.

Разрешено:

- свободное использование;
- копирование и передача;
- включение в бесплатные обучающие программы;
- адаптация под личные нужды.

Запрещено:

- продажа в любом виде;
- размещение в платных курсах и вебинарах;
- любая форма монетизации;
- выдача материала за коммерческую собственность.

Статус: Freeware / Только образование / Коммерческое использование запрещено.



NIKA
SOFTWARE